Charles University in Prague
Faculty of Mathematics and Physics

Diploma Thesis

Jan Staněk

DoS a DDoS útoky na SIP protokol

DoS and DDoS attacks on the SIP protocol

Department of Software Engineering

Thesis supervisor: RNDr. Ing. Jiří Peterka
Study Programme: Informatics, Software Systems

2010

I would like to thank my thesis supervisor, Mr. Peterka, for his valuable remarks and comments, which I used in improving this thesis. I would also like to thank my family for their support and my friends for their understanding in times when I did not have much time for them. My gratitude also goes to my colleagues at work for their overall support and some useful hints.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 8.12.2010 Jan Staněk

Contents

Název práce: DoS a DDoS útoky na SIP protokol
Autor: Jan Staněk
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Ing. Jiří Peterka
e-mail vedoucího: peterka@ksi.mff.cuni.cz

Abstrakt: Cílem této diplomové práce je seznámit se s protokolem SIP a s problematikou možných útoků na tento protokol, zejména se zaměřením na útoky typu DoS a DDoS. Práce se soustředí na podrobnou klasifikaci útoků, možnosti a způsoby generování těchto útoků a metodiky obrany proti nim. Velký důraz je kladen zejména na útoky záplavového typu (floods), neboť jejich generování je velmi snadné a zařízení, která jsou typickou součástí SIP architektury, jsou velmi náchylná vůči těmto útokům. Součástí práce jsou také pilotní implementace některých zásadních myšlenek týkajících se jak generování útoků, tak obrany proti nim a jejich praktické testování v simulovaném SIP prostředí.

Klíčová slova: SIP, DoS, DDoS, IP telefonie

Title: DoS and DDoS attacks on the SIP protocol
Author: Jan Staněk
Department: Department of Software Engineering
Supervisor: RNDr. Ing. Jiří Peterka
Supervisor's e-mail address: peterka@ksi.mff.cuni.cz

Abstract: The aim of this diploma thesis is to get accustomed with the SIP protocol and with the problematics of attacks targeting this protocol, with the emphasis on DoS and DDoS attacks. The thesis focuses on detailed classification of the attacks, possibilities and forms of generation of the attacks and methodics of defense against them. The attacks of the flood type are especially stressed because they are easily generated and the SIP components are very prone to these attacks. Prototype implementations of the most important ideas concerning attack generation and protection against these attacks are also part of this thesis. Practical tests of the implementations performed in a simulated SIP environment are also included.

Keywords: SIP, DoS, DDoS, IP telephony

Chapter 1

Introduction

Voice over Internet Protocol (VoIP) is becoming very popular in the last few years. It is generally much cheaper than the Public Switched Telephone Network (PSTN) and offers many additional services. Because of the massive deployment of VoIP, used technologies are developing quite fast and the development process is oriented mainly on the services and functionalities it provides. Security has been considered less significant for a long time. The most commonly used signaling protocol in VoIP is the Session Initiation Protocol (SIP). Although the first standardization of SIP was done eleven years ago and great many changes and modifications has been proposed and developed in the field of its security since then, still it is very common that deployed VoIP environments based upon SIP use only the basic security mechanisms which offer almost no protection against a wide variety of SIP-specific attacks.

Because the area of SIP security is quite extensive and the thesis cannot cover all the aspects of it without becoming unmanageably big, I decided to focus on the SIP-specific attacks resulting in a Denial of Service (DoS) and its more complex distributed versions (DDoS) and defense against them. To preserve the completeness of the thesis, the types of attacks targeting used and supportive technologies are briefly mentioned in chapter *Problem Analysis*, but are not being furthermore investigated. I have chosen the area of DoS and DDoS attacks because they are fatal for the whole SIP service and there is no generally accepted way of how to defend against them yet. Also, the SIP servers are usually very prone to these attacks.

This thesis does not aim to provide an implementation of a perfect defense solution that would be able to mitigate all types of SIP-specific DoS and DDoS attacks (that is something unachievable because the attackers are very likely to come with new original exploits every time a new defense solution is proposed). My main concern is to show that there are some very crucial security aspects

that should be addressed by security researchers and SIP developers and users. I will demonstrate how simple it is for an attacker to produce attacks that are fatal for unprotected SIP environments and I will also provide some ideas that can be used for mitigation of these attacks.

The structure of the thesis is organized as follows. The fundamentals of SIP, DoS and DDoS attacks are available in chapter *Basic Concepts*. The classification of attacks according to its different patterns and resulting impact on the SIP environment is described in chapter *Problem Analysis*. My survey of the existing defense mechanisms against DoS and DDoS attacks and a brief description of their functionality is available in this chapter too. To test how complicated it is for an attacker to create a successful attack against a SIP sever, I have used some publicly available tools to simulate such attacks in chapter *Attack Realization*. In the chapter *SIPp-DD: A complex SIP DoS and DDoS attack simulation tool*, I introduce a SIP DDoS attack simulation tool called SIPp-DD, which I created by modifying a very popular SIPp call generator tool [1], and address the typical characteristics of a DDoS flood attack. A new approach to defense against flood type DDoS attacks, which I developed, is described in chapter *New Defense Solutions*. To prove the usability of my new approach, I have implemented a prototype and tested it practically in DoS and DDoS flood simulations. The details about the tests that were done along with their results are available in chapter *Tests*. Some ideas for future research along with the summary of the work described in the thesis are in chapter *Conclusion*.

Chapter 2

Basic Concepts

2.1 Voice over Internet Protocol (VoIP)

VoIP was formerly a term for protocols and mechanisms used to transport voice communications over the Internet Protocol. Nowadays this term is used in more general meaning and covers the area of almost any multimedia transport over IP. The most typical extension is adding a visual information to the voice communication which results in a videocall. Other common extensions are text messages, presence notification etc. Next to VoIP, terms Internet telephony or IP telephony are often used and have the same meaning. Note that the term Internet telephony is somewhat misleading since the general idea does not necessarily use the public Internet. VoIP solutions are very often used in private networks to satisfy the needs of internal company communication.

In the early stage of VoIP development, the H.323 standard was created as a recommendation from the International Telecommunication Union (ITU). This standard is quite complex and addresses all aspects of the VoIP using different protocols. Some examples of these protocols are

H.225.0 Call signalization protocol

H.245 Control protocol for multimedia communication

H.235 Protocol providing security of multimedia and signaling

H.460 Protocol for optional extensions

RTP Real-time Transfer Protocol for media delivery

These were just some examples of wide variety of protocols defined in the H.323 standard. From todays perspective I can say that H.323 was a good start but

it was rather complex and therefore complicated to be fully implemented for smaller employments. Some simplification was necessary and therefore only two core parts of the VoIP service were identified – signaling and media delivery. The media delivery was quite satisfactorily provided by the RTP and there was a need for a simple signaling protocol.

Many protocols were developed to fill in the free position of the signaling protocol. The Session Initiation Protocol (SIP) surpassed the other proprietary protocols and became the most easily adopted and quite fast also the most often used signaling protocol in VoIP. Nowadays, the VoIP environments are usually based upon the combination of SIP as the signaling protocol and RTP as the media delivery protocol. Because of this situation, I have chosen SIP for my security research. Note that the mentioned protocols are not all protocols used in VoIP. There are hundreds of other protocols focusing on different aspects of the VoIP but these are out of the scope of this thesis. If you are interested in VoIP in general, I can recommend you the book written by Camp [2]. The H.323 standards [3] are also a very good source of complex information about the VoIP.

2.2 Session Initiation Protocol (SIP)

2.2.1 Introduction

SIP is a signaling protocol widely used in VoIP. It offers the functionality of session manipulation (e.g. session creation, modification and termination). In terms of voice communication the session corresponds to a call. Next to unicast (classic call between two participants), SIP also provides multicast (conference call). SIP can also handle more than one media stream and can therefore provide parallel media delivery for one session (for example a videocall). SIP was standardized by the Internet Engineering Task Force (IETF) in RFC 2543 [4] in 1999 and revised in RFC 3261 [5] in 2002. In the following sections of this chapter I briefly describe the protocol architecture and point out some security concerns. The details of concrete parts of the protocol necessary for better understanding of different attacks investigated later in this thesis are discussed in parts regarded to these attacks. For more detailed description of the whole protocol please refer to the RFC 3261 [5].

2.2.2 Architecture

SIP architecture is based upon User Agents (UA). User Agent is a network element capable of creating and sending as well as receiving and processing SIP messages. User Agent can play one of two roles in the session. It can be either

a User Agent Client (UAC) which sends SIP requests or a User Agent Server (UAS) that receives the requests and generates the responses. Common UAs are software and hardware SIP phones. The identification is done using the Uniform Resource Identifiers (URI). Each UA as well as each other resource has its own unique identifier.

Next to UAs, there are also other network elements extending the possibilities of SIP. Even though SIP can be used as a peer-to-peer protocol – no central server, the session is created directly between the endpoints – it is a very impractical setup for bigger deployments since it offers no central management. Therefore some server elements were also proposed in the SIP architecture. These server elements are: proxy server, registrar and redirect server. Proxy server works the same way as proxy servers in other architectures – it forwards the requests and responses among other network elements and is able to change some parts of the communication according to its setup. Proxy server can be either stateless or stateful depending on whether it does or does not keep the information about the state of the session. Registrar is a server which accepts the SIP REGISTER requests and provides the information gathered from them to location services. Redirect server is a server which sends the SIP redirection messages and provides the functionality of accessing external domains. These three server elements are usually implemented in a form of one component referred generally as the SIP server. From now on, I will be using the term SIP server for the component integrating all these three server elements in itself.

There are also other network elements next to the UAs and server elements. These are generally gateways providing interconnection with different types of other networks such as PSTN and mobile networks or Session Border Controllers (SBC) used as "guards" on the borders of a SIP environment. These elements have very specific functionalities and I will not look further into them now.

SIP standard defines three types of transport usable in SIP communication. The first two are classic UDP and TCP, the third is encrypted transport using TLS. The default port value for the SIP service using UDP or TCP transport is 5060. The secure version of SIP using TLS encryption is usually available on the port 5061. Even though it is possible to change the default values, it is done very rarely and I have even encountered SIP phones which were unable to change the destination port.

2.2.3 Messages

SIP messages are divided into two groups – requests and responses. The typical message exchange process can be described as follows

1. UAC generates a request and sends it to UAS

2. UAS processes the request and sends a response to UAC

3. UAC processes the response and may continue by generating a new request

Both types of SIP messages consist of header and body. The header has precisely defined structure and is slightly different for requests and responses. Let us have a look at the requests first.

Requests

The most important part of the SIP request header is the first line. It begins with a keyword specifying the type of the request. The keyword is followed by an URI of the requested participating UAC and the URI is followed by the specification of SIP version. As you can see, the first line contains all the basic information – said in human language it describes what is requested from whom. The rest of the header specifies additional information like who should be contacted, what is the contents of the message etc. An example SIP request header looks as follows

```
INVITE sip:userA@domain.com SIP/2.0
Via: SIP/2.0/UDP pc1.domain.com
Max-Forward: 70
To: ‘‘userA’’ <sip:userA@domain.com >
From: ‘‘userB’’ <sip:userB@domain.com >;tag=123
Call-ID: 1234567890@171.1.1.1
CSeq: 1 INVITE
Contact: <sip:userB@pc1.domain.com >
Content-Type: application/sdp
Content-Length: 100
```

There are six basic SIP requests which must be implemented in each SIP server plus eight requests added as an extension. The implementation of extensions is optional. The basic requests are (in alphabetical order)

ACK Confirms that the client has received a final response to an INVITE request.

BYE Terminates a call.

CANCEL Cancels any pending request.

INVITE Indicates that a client is being invited to participate in a call session.

OPTIONS Queries the capabilities of the SIP server.

REGISTER Registers the address listed in the To header field with a SIP server.

The extensions are INFO, MESSAGE, NOTIFY, PRACK, PUBLISH, REFER, SUBSCRIBE and UPDATE. More details about the extension requests as well as about all the possible header fields are available in the SIP reference manual [6].

Responses

The response header is very similar to the request header. The first line is again the most important one and its structure is `sip_version sip_response_code sip_response_phrase`. A typical first line of a response header look as follows

```
SIP/2.0 200 OK
```

The rest of the header is the same as in the request header. The response codes are very similar to the HTTP response codes. The response phrases are as short as possible to remain understandable. There are six groups of SIP responses defined as follows (x represents any digit 0-9)

1xx Provisional – request received, continuing to process the request;

2xx Success – the action was successfully received, understood, and accepted;

3xx Redirection – further action needs to be taken in order to complete the request;

4xx Client Error – the request contains bad syntax or cannot be fulfilled at this server;

5xx Server Error – the server failed to fulfill an apparently valid request;

6xx Global Failure – the request cannot be fulfilled at any server.

Each group of response codes contain from few to many concrete response codes. Each response code has a corresponding response phrase and meaning. For more details about the response codes refer to the RFC 3261 [5].

2.2.4 Brief security overview

SIP was firstly standardized in the document RFC 2543 [4] and only minor security considerations were taken into account. The biggest problem that was not solved was that some important parts of the SIP message header could not be encrypted since it contained the information necessary for successful delivery of the

message and thus it had to remain in clear text. For the rest of the SIP message that can be encrypted, the standard specifies that the implementations SHOULD be able to use the PGP encryption scheme and MAY use other schemes too. The implementations SHOULD be able to use basic and digest authentication and the requests MAY require authorization. Briefly summarized, the standard offers possibilities of how to provide some security (but hardly sufficient since there are still problems with message integrity etc.) for SIP but since it does not enforce usage of any of these mechanisms, the proprietary implementations often did not contain them.

The security situation for SIP changed in 2002 when the document RFC 3261 [5] was released. This revised standard of SIP introduced SIPS, a secure version of SIP. Using SIPS URI instead of the classic SIP one ensures that the SIP communication will use an encrypted transport mechanism. TLS, IPSec and S/MIME were proposed as possible encryption mechanisms. Using one of these mechanisms, all the SIP communication could be encrypted and therefore its completeness and integrity should be guaranteed. Well, should be, if you use one. The description of the methodics of how to use the encryption mechanisms for SIP is quite detailed in the standard but once again the standard states that this functionality is optional, MAY be implemented and MAY be used.

Next to the already mentioned security considerations there are also the weaknesses which SIP inherits from the lower layers it uses (e.g. transport layer, internet layer etc.) and from other standardized protocols it might use (like DNS). I will have a closer look at these issues later in this thesis.

Briefly summarized, the problem nowadays is not the lack of mechanisms usable to secure the SIP environments but the lack of their support in many implementations of SIP components and the indolence of system administrators who very often use the most simple deployment – UDP transport without any proper defense mechanisms. Therefore, I will focus on demonstrating how dangerous this approach is and how easy it is to deploy a devastating attack against such environments as well as how relatively easy one can create at least basic defense capable of mitigating plenty of these attacks even without the usage of advanced security mechanisms proposed in the SIP standard.

2.3 Denial of Service (DoS) Attack

Denial of Service attack is a general term that is used for an attack that makes the targeted resource/service unavailable to its legitimate users. Taking this definition, you can say that almost anything is a DoS attack. If you take a a cup of coffee and spill it to your colleagues keyboard making it useless, you have just successfully done a DoS attack. Because of this somewhat generic definition, the

term DoS attack is commonly used for more specific type of network attacks targeting servers providing public services (this very often means Web servers providing HTTP services). Some authors also use the term DoS attack for flood attacks (e.g. attacks using huge numbers of packets sent to the target server or network, which results in a congestion). That is however a bad usage of this term since the flood attack is only one of many subtypes of the DoS attack.

In this thesis I will use the more specific meaning – software attacks against network services. I will completely omit the physical DoS attacks. Not because they are not important (you should always consider them when implementing a security politics) but because there is no implementable software defense against them.

2.4 Distributed Denial of Service (DDoS) Attack

Distributed Denial of Service attack is a subtype of DoS attack that has more than one source. Using the model situation from the previous definition, a coffee spilling DoS attack will become a DDoS attack when your other colleague will take his cup of coffee and join you in the process of keyboard drowning. Well, let us finish with these funny models and keep focused on the real situations. A typical DDoS attack is a flood attack that comes from many different sources (e.g. there is no single source generating the load congesting the target server but there are many sources flooding the target server).

DDoS attacks are often produced by so called botnets. Botnet is a name for a set of computers that are controlled by the attacker. The computers in a botnet are typically computers owned by common users or corporations which were infected by a worm that gives the attacker the possibility to control these machines. Because of this principle, it is quite hard to get rid of botnets since the owners of the machines usually do not know that their computers are part of an organized malicious network.

Defense against DDoS attacks is generally more complicated than defense against DoS attacks since the typical approach of traffic limiting combined with source blocking is useless. I will address these differences and defense solutions in more detail later in this thesis.

Chapter 3

Problem Analysis

3.1 Attacks on the VolP Environment – General Classification

Before I will get to the specific DoS and DDoS attacks on the SIP, I would like to provide you with the overall illustration of attacks against VoIP environments so that you become familiar with the bigger picture. There are numerous attack types that an attacker can use targeting a VoIP environment. The most complex classification of these attacks is available in the document from the VOIPSA organization called *VoIP Security and Privacy Threat Taxonomy* [7]. However, this document is very complex and its structure might be confusing for the readers. Therefore I have created my own classification of the attacks. From my experiences, there are two main aspects of each attack – target service and impact. By target service I mean the service which is exploited by the attack and by impact I mean the result of the attack on the VoIP environment (or its owner). From this point of view, the classification can be done as follows.

According to Target Service

Signaling protocol Attacks against the signaling protocol itself are used for various purposes - identity theft, traffic rerouting, service disruption etc.

Media delivery protocol Attacks against the media delivery protocol aim mainly at traffic interception and modification.

Underlying layers Attacks that are not targeting VoIP directly but are targeting services of lower layers (Transport layer, Internet layer or Link layer) instead.

Additional used services Attacks on the application layer that are not targeting VoIP directly but instead the services that VoIP uses. These are, among others, DNS for name resolving and databases for billing and user management.

According to Impact

Service disruption Attacks aim at disruption of the VoIP service making it unavailable for the legitimate clients.

Billing exploitation Attacks aim at financial part of the service – using the service for free, making a financial loss for the service provider or even direct self-enrichment (initiating calls to owned premium services from the target VoIP environment).

Information sniffing Attacks aim at extraction of private information from the signaling messages or from the calls.

Disinformation Attacks aim at tricking the users of the service to believe what the attacker wants. Often combined with information sniffing.

I have not provided a detailed description of all attack types intentionally, because there are hundreds of different attacks against VoIP and it is not in scope of this thesis to investigate all of these. Instead, I have tried to cover the attack space from the top view to give you an overview of the whole attack space and I have provided you with a simple methodics how to divide the attacks into groups according to their most significant aspects. From now on, I will focus primarily on service disruption attacks targeting the signaling protocol (e.g. in my case more specifically DoS and DDoS attacks on the SIP protocol).

3.2 Service Disruption Attacks on the SIP Protocol – Classification

Similarly to the classification which I proposed in the previous section, I have created my own classification for this type of attacks too. This classification should help me in keeping the rest of the thesis tidy – I want you to always understand the main aspects of the attack which I try to simulate or mitigate in corresponding parts. I have identified three main aspects for each attack. These aspects are: methodics, target and distributiveness of the attack. Because these aspects are essential, let us have a closer look at each of them.

3.2.1 Methodics of the Attack

By methodics of the attack I mean the principle using which the attacker disrupts the service. There are three most often used principles – malformed message, flooding attacks and message misuse.

Malformed Message

Malformed message (often also called killer message, malformed packet or killer packet) is a methodics using a modified version of legitimate SIP message which, when processed by a SIP server or a SIP client, results in a critical failure of the component and a consequent restart or reboot is necessary. Because there are hundreds of implementations of SIP servers and SIP clients available on the Internet, there are also numerous exploitable holes in these applications (since there is no such thing as an infallible programmer). Even if the parser of the SIP component is written flawlessly to process every type of well-formed SIP message, there is an infinite number of non-well-formed messages which might produce the failure and noone can take into consideration all of these messages when programming a SIP component.

Flooding Attacks

Flooding attacks (floods) are based upon usage of high numbers of SIP messages which leads to the congestion of the target SIP component (server or client) forcing it to consume all of its resources of some kind (most often memory or CPU usage). Because SIP is a signaling protocol, the components are generally not optimized to process high numbers of messages per second since there is no need for it when processing common legitimate traffic. When the attacker exploits this fact and periodically generates high numbers of SIP messages and sends them to the target, the target becomes congested in a few seconds and becomes unavailable for the legitimate traffic. Floods are generally not designed to crash the SIP component and after the attack ends, the component should return to normal state. However, as I have proved in some of my tests, this highly depends on the implementation of the component and on the resource consumption produced by the attack. I have encountered for example a situation where all the RAM was consumed by the SIP server during the attack and it was not freed after the attack ended, so restart of the SIP server was necessary.

There is also one special subtype of flood attacks called the reflection attack (sometimes also called mirror attack). The basic principle is the same as for the common flood attack (sending high number of messages) but the attack is masked such that the real attacker seems not to be involved in the attack at

all. He tricks legitimate SIP components into being the originators of the attack. The process of a reflection attack works as follows:

- The attacker periodically sends a high number of SIP requests to different SIP servers. The messages have fake source IP address (using the spoofing mechanism). As the fake source IP address, the address of the targeted SIP component to be targeted by an attack is used.
- The SIP servers generate responses to each of the requests and send it directly to the targeted SIP component (since they believe that it was the originator of the request).

This way, the targeted SIP component is flooded. Additionally, the attack is in fact a distributed one even though the original attacker needs only one machine and even if there are good defense mechanisms deployed on the targeted SIP component, the attacker will not be blocked from reaching it (moreover, the misused legitimate servers might become blocked from reaching the targeted SIP component).

Message Misuse

Message misuse attacks use well-formed SIP messages injected into legitimate SIP traffic with the effect of disrupting the service or preventing the users from reaching the service. The difference from the previous two attack types is that message misuse attacks do not aim to crash or congest the SIP components. They are crafted specifically to hijack the SIP communication itself and teardown the sessions in progress or prevent new sessions from being created. A typical example of a message misuse attack type is a BYE attack – an attacker sends a BYE message specifically crafted to target a session in progress to one of the participants in the session. This participant expects the other participant to be the originator of the BYE message and ends the session on his side. Very similar to BYE attack is CANCEL attack that targets the sessions which are in a phase of preparation (the connection was not yet formed, only initial request for it was sent).

For all attacks of the message misuse type, the attacker needs additional information about the SIP traffic (such as call IDs etc.) and a possibility to inject his own crafted SIP messages. Therefore the attacks of this type are usually deployed using MitM (Man-in-the-Middle) mechanisms where the attacker situates himself into the center of the SIP traffic (by overtaking the SIP server, exploiting SIP redirect mechanism or similar method). Because of the need of additional information and access, the message misuse attacks are more difficult to deploy than flooding and malformed message ones and I will not focus on them much in

this thesis. If you are interested in these attacks, I can recommend you a book written by Endler and Collier [8].

3.2.2 Target of the Attack

In a SIP environment, you can target two main components:

SIP server The most important component of the environment is also most often target of the attackers because disabling this component means effectively disrupting the whole SIP service.

SIP client If the attacker does not want to disable the whole service but only to make it inaccessible for one user, he usually targets the corresponding SIP client.

Even though this classification looks simple, there is one important fact to be considered and that is accessibility of these components. SIP servers are usually situated on the border of private networks because it is desired for them to be accessible from both the private network and the Internet (users want not only to be able to call outside but also to be called from outside). This means that if the attacker does not have access to the private network, it is simpler for him to target the SIP server. Therefore protecting the server has much higher priority since the client can be (almost always) hidden in a non-accessible environment and therefore any attack targeting the client must go through the SIP server.

Because the SIP server is more complex than a SIP client and it is more often targeted by the attacks, I focus on attacking and defending the SIP servers in my tests. However, since both SIP components are similar, the approaches to attack and defense does not differ much and every proposed attack strategy and defense solution can be applied to both components with minor changes.

3.2.3 Distributiveness of the Attack

The basic idea of the distributiveness of the attack is very simple – either the attack is distributed (has more than one source) or it is not. If you have a closer look, however, you will find that the problem is a little bit more complicated. First, one has to specify what is a source. In this thesis, I define the source as the source IP address from which the packet containing the SIP message arrived (generally, I should use the combination of source IP address and port, however, the port is not very important in this case). In the terms of distributiveness of the attack, I do not care about the actual contents of the SIP message (e.g. if the attacker for example uses different usernames in the *From* field in one flood,

the flood is considered to be a DoS flood if the source IP address for all packets, which are part of the attack, remains the same).

Another problem are spoofed addresses. If the attacker spoofs the source IP addresses in an attack and sends the packets from only one machine but with spoofed different addresses, I consider the attack to be a DDoS. This consideration is made due to the fact that the server receiving these packets will consider them to be from different sources and will process them that way (which is very important for example for simple DoS defense mechanisms based upon source filtering).

One also has to consider the situation where the originator of the SIP message is a legitimate client or server and the message is modified before reaching its target by malicious application inserted to the SIP flow by an attacker. This type of attack is called Man-in-the-Middle attack (MitM) and it is hard to say whether it is to be considered a DoS attack or a DDoS attack. If the MitM application changes only packets of one flow (meaning the connection between two endpoints) the attack can be considered a DoS. If, however, the application resides before the SIP server and changes all incoming SIP messages coming from different clients, the attack can be considered DDoS.

Attacks consisting of only one packet (usually malformed packet) are considered to be DoS attacks, even if the source address is spoofed.

3.3 Defense Solutions – Theoretical Principles

In this section, SIP DoS and DDoS attacks are divided into a few groups and some ideas of defense solutions against attacks from each of these groups are proposed. The proposed solutions are weighted from various angles and their strengths and weaknesses are discussed. Some of the proposed solutions are implemented as prototypes in the latter part of this thesis and the results of practical experiments with them are also enclosed. Note that I have not invented these solutions – mostly all of them are publicly known and I have only summarized them and added a few ideas that improve them or change them a bit here and there. New defense solutions proposed by me are described in chapter *New Defense Solutions*

When considering defense approaches against SIP DoS and DDoS attacks, I can divide the attacks into groups based upon the previously stated main aspects. The aspect of the target is not very important for the defense solution design – whether the attack is deployed against a SIP server or against a SIP client, the countermeasures will be very similar and will only differ in the setup. On the other hand the aspect of distributiveness is of utmost importance as well as the aspect of methodics. Therefore I divide the attacks into four groups based upon these aspects and have a closer look at each of these groups independently.

3.3.1 Malformed Message DoS Attack

This type of attack is very common. Usually a tool with a predefined set of malformed SIP messages is used and its only functionality is to successively send the messages from the set to the target SIP server and monitor whether or not the processing of the message leads to the situation where the server is unable to process SIP messages (crash, critical failure, infinite loop etc.). The attack rate is usually low, only a few packets per second, so that the attack will not become a flood DoS, which is easily detected and mitigated.

Using the knowledge about the common attack process, one can design a few different defense solutions. The first solution I considered is based upon message preprocessing. One can use an application which will process all the SIP messages before they get to the SIP server and check whether they are well-formed. If they are, they are forwarded to the SIP server. If they are not, they are discarded. This solution is very straightforward in theory but has some drawbacks when you focus on the real situation. The biggest drawback is the complexity of the checking process which will be very time and resource consuming and might lead to unwanted delays in the SIP communication. The other drawback is the expectation that the SIP server is capable of processing all well-formed SIP messages, which is not true in many cases. The advantage of this solution is that no human interaction is necessary.

The second solution uses post-processing of the log produced by the SIP server. Every good implementation of a SIP server is able to log the SIP messages it receives. This way, if the server gets to an inconsistent state and becomes unable to process SIP messages, the last logged message is very probably the problematic one. Using the log information, the server can be patched to be able to process such messages in the future. The biggest drawback of this approach is that human intervention is necessary to patch the server and also that the first attack and each consequent attack of the same type succeeds in disabling the server before the patch is created and applied. Another disadvantage is that if the server is targeted by a flood attack, the logging mechanism will consume additional resources and might worsen the impact of such attack. The advantage of this approach is that with each detected attack, the SIP server implementation becomes less vulnerable to attacks of similar type.

The third solution uses post-processing of the log produced by the SIP server. It requires the server to be able to log every message that was marked as non-well-formed during the processing phase (which is an integrated functionality in many SIP servers). Because the tools used by an attacker are generally using a set of malformed messages, one might suppose that the server will be able to withstand a few of the first ones. If so, these tries will be logged. The only functionality of this defense solution will be to periodically check the log and if a

sequence of messages of length n, where n is a reasonably small number, coming from the same source appears in the log then the source will be blocked from sending any more messages (this can be done for example in form of a firewall block rule). The advantage of this solution is that it is easy to implement and effective if the server withstands the first n malformed messages generated by the tool used by the attacker. The obvious disadvantage is the situation when the critical message disabling the server comes in the set of the first n malformed messages.

The fourth solution is similar to the first one. It uses preprocessing of the messages but instead of checking the validity of SIP messages, it uses regular expressions to check the patterns of known harmful ones. The messages marked as harmful are dropped, the others are forwarded to the SIP server. This approach has the advantage that it does not consume so much resources as the first one and is able to drop the malicious requests. The disadvantage of this approach is the fact that someone has to create the initial set of harmful patterns and this set can never be complete (e.g. there may always appear a message that will disable the server upon processing and is not matched to any known harmful pattern).

The fifth solution combines the fourth solution with the second one. The principle is the same as in the fourth solution but it adds a mechanism to periodically check whether the SIP server is OK and if not, then it restarts the server and computes a harmful pattern from the last logged SIP message which it adds to the set of known harmful patterns. This solution is the best of the proposed solutions and its biggest advantages are the facts that it does not need human intervention and is able to learn new harmful patterns automatically. However, there still remains the disadvantage that each attack that contains a harmful message, which is not in the harmful pattern set of the defense solution, will disable the server. The practical implementation of the logic creating the harmful patterns from logged messages would also be quite complicated task requiring a lot of research and invention.

Generally, it is a very good idea to test the implementation of the SIP component you plan to use in your environment apriori to its deployment. Some details concerning the tools that might be used for such testing are available in the chapter *Attack Realization*.

3.3.2 Malformed Message DDoS Attack

This type of attack is quite uncommon and I have not encountered a single one nor have I found any tool designed to produce such attack during my research. Considering the consequences, however, I have found out that this attack has

its significance. If the server targeted by this attack has active defense solution described for the previous attack type as the third one, this defense will be bypassed and totally ineffective. Why? Because the defense mechanism will try to find logged sequence of messages from the same source, which, thanks to the distributiveness of this attack, will not be present.

Otherwise, this attack is very similar to the malformed message DoS attack and other defense solutions described for that type (with the exception of the third one, as mentioned) are usable against this type too.

3.3.3 Flooding DoS Attack

A very common type of attack. The attacker can choose from various tools which are freely available on the Internet and the deployment of the actual attack is often only a matter of a basic setup of the mentioned tools that will take only a few minutes to even an unexperienced computer user. The functionality of the tool itself is also trivial – the tool periodically sends already prepared SIP messages to the specified SIP URI using specified message per second rate. Therefore the attacker only sets the URI of the target, message rate per second, type of the SIP message to be sent and starts the attack. Setting message rate to a few thousand messages per second and choosing the right SIP message type is enough to congest every freely available SIP server (and supposedly even the paid ones).

With the knowledge about the common attack process, one is able to design a few defense solutions right away. Thankfully, the attack which is most easily created is also most easily mitigated and my proposed solutions are both efficient and easy to implement.

The first solution uses an integrated firewall that is a part of any operating system (even though I did not try this on Windows). One has to compose a simple blocking rule – if the packet count from one source IP address on the SIP port exceeds the predefined limit, the source IP address is blocked for a defined time. The only thing that one needs to setup are the three limits – the time period, the threshold of accepted packets per time period and the time for which the source IP address will be blocked if its SIP traffic exceeds the threshold. These values differ for different SIP environments since in every SIP environment, there is another level of common SIP traffic (depending on the number of clients and common usage). It is a good idea to tailor these values specifically for the target environment but if you do not have time or resources to monitor the common traffic and do statistic assumptions, the threshold of 50 messages per second for one source is an acceptable choice. I cannot imagine an environment where the legitimate client will exceed this threshold and also any good implementation of

a SIP server is able to process a few hundreds of messages per second. As for the time for which the potential attacker will be blocked, I have used 10 minutes for test purposes but it does not have a high importance.

Even though the first solution is very simple to implement and will mitigate the majority of SIP DoS flood attacks, you should not take it as a sufficient defense against SIP threats. As I have already showed in the previous parts (and will show in some of the following ones), there are many more dangerous threats and even this simple solution itself can be exploited to deploy a very specific type of attack. Also there are some environments where this simple solution cannot be deployed for example because there is a need to use high amount of SIP INFO messages to deliver additional information.

The second solution is slightly more complex than the first one but has better results and is usable in some special cases where the first solution is not (for example the already mentioned use of many SIP INFO messages). This solution works on the same principle as the first one but makes difference between different SIP messages. It does not count the overall number of SIP packets arriving at the server but it checks the first few bytes of the packets to find out which SIP message is contained in the packet and has defined different limits for different types of SIP messages. This way, you can define limits only for some types of SIP messages (for example the ones which consume most resources to be processed) and do not care about the others. The advantages of this solution over the first one are obvious, the disadvantage is a slightly more complicated implementation (setup) of such solution and higher resource consumption of the solution itself.

The third solution is different from the previous two and uses post-processing based upon the log created by the SIP server. Again, the principle is quite straightforward. The SIP server is set up to log processed SIP messages. The defense application periodically checks this log and if it encounters more than n messages from one source IP address it sends a blocking rule to the firewall to block this source IP address. Again, one has to choose the value for n and the time for how long the source IP address will be blocked. The advantage of this solution is that the SIP messages are delivered directly to the SIP server without any delay and that one can setup the logging policy such that only some types of SIP messages are logged. The disadvantage of this solution is that if the attack is very strong (the message rate per second in the attack is very high), this solution might react slowly and the server may get to a congested state for a short time.

All the three mentioned defense solutions were implemented using only publicly available tools for testing purposes. Details about the tools that were used and some remarks about the solutions are enclosed in Appendix B.

3.3.4 Flooding DDoS Attack

This type of attack is very interesting and even though it looks like only an advanced version of the SIP DoS flood attack, the defense mechanisms used to mitigate SIP DoS flood attack are ineffective against SIP DDoS flood and can be even exploited to produce a new type of attack. From the characteristic of the attack is obvious that the approach of source IP address blocking is useless in this case because the attacker can always distribute the attack load among the bots such that it will not trigger the thresholds. The deployment of a SIP DDoS flood attack is much more complicated since the attacker needs a botnet which he can use to deploy an attack. There is also another way than using a botnet and that is using a source IP address spoofing mechanism. This mechanism, however, is not trivial to implement and even if one implements it, there still remains the problem of how to get the packets with spoofed source IP addresses to the target (since routers can check the origin of the packet, NATting is a problem etc.).

Because of the described characteristics, I have considered this attack type to be very interesting for research and I have developed my own attack generator (capable of IP address spoofing) and two original defense solution mechanisms. The details concerning design, implementation and many notable side-revelations which appeared during my research are available in the next chapters of this thesis.

3.4 Related Work

Even though this thesis should not be a background research of existing projects concerning the problematics of DoS and DDoS attacks on the SIP protocol, it would be a waste of time trying to re-discover something that was already discovered or re-implement something that was already implemented. Therefore I have done a survey of publications with similar topic before I began my research in this field and I would like to share brief summaries of the projects I have discovered. I would like to denote that the summaries I created may contain facts which I derived from my observations and these facts may be subjective to my point of view. By these summarizations I do not want to judge or evaluate the mentioned works, I just try to point out things which I consider important for my own work.

3.4.1 SIPp

SIPp is an open source traffic generator that was designed specifically for testing purposes. Sipp is capable of simulation of both UAC and UAS and can also

generate both signaling and media traffic. The original source code of SIPp was written by Richard Gayraud and modified by Olivier Jacques, but thanks to its rapidly growing popularity, it quickly became a community tool and many authors joined its development. At the beginning, the popularity of SIPp came from the fact, that it was quite easy to use. The original simple command line interface was intuitive and provided all the necessary functionality for generation of SIP testing communication. During the development, the CLI was additionally extended by usage of XML files specifying the SIP communication scenarios and these scenarios were extended by the possibility of insertion of data from external CSV files.

Another big advantage of SIPp is that it was (and still is) open source, written in C++. Therefore every user can easily implement any functionality according to his needs. Many patches for SIPp were created, extending its functionality in many ways. Thankfully, the maintainers of SIPp did not allow the tool to become a multi-branched unmanageable monster and restricted the way the patches were being adopted by the stable version of SIPp. The patches that did not make it to the stable version were made accessible in the project webpage for anyone to use or improve. Thanks to that, SIPp remained relatively simple and easy to use but there are plenty of extensions available. One of these extensions contains also usage of a RAW socket, however, the patch was made for a very obsoleted version and was not maintained. Even though, it was a good starting point for me when I began with implementation of my attack simulation tool SIPp-DD (details about it are available later in the thesis).

If you are interested in SIPp, quite a lot of information can be found at its webpage [1]. You are encouraged to become familiar with SIPp since it is a very good SIP testing tool and moreover, once you learn how to work with SIPp, working with SIPp-DD will make you no problem at all.

3.4.2 The SNOCER Project

SNOCER is an abbreviation for Low Cost Tools for Secure and Highly Available VoIP Communication Services and it was a joint project of both academic and business institutions supported by EU funds. The project was held in 24 months between 1st November 2004 and 1st November 2006. The goal of the project was already mentioned in its name – the project aimed at creation of a reliable and robust VoIP infrastructure resistant to different VoIP-specific attacks.

The outcome of this project are two technical reports which are available at the project web page [9] and a few conference papers, also available at the web page. The first technical report contains basic information about security mechanisms integrated in SIP and the threats to the SIP environment from var-

ious sources. There is also a part about reaching reliability of the VoIP service using duplication and delegation mechanisms. The second technical report describes the most common attacks against VoIP environment and offers a design of defense architecture capable of their detection and mitigation. The conference papers describe various parts of the project which are in more or less detail described in the already mentioned technical reports.

The technical reports are good, however, the project goal was chosen quite generally and therefore many aspects of the VoIP security were not taken into account. It is especially crucial that DDoS attacks were not thoroughly considered and the proposed detection and defense mechanisms are not able to handle these attacks even though these attacks are one of the biggest threats to VoIP infrastructure. Otherwise, the proposed defense architecture seems very interesting and it is a pity that its bigger part remained only in a form of a theoretical design. I have taken inspiration in this architecture when I was designing my own defense solution against DoS and DDoS attacks on SIP.

3.4.3 SecSip

SecSip is a stateful firewall for SIP-based networks designed and implemented by researchers from LORIA laboratory in France. The description of SecSip together with the implementation and some conference papers written about it are available on the project website [10].

The idea of an application firewall is very interesting and using statefullness to detect anomalies in SIP transactions is an innovative idea. The SecSip language developed for easy construction of rules created for the SecSip firewall is intuitive and easily adopted and that makes the SecSip implementation suitable for use in real SIP environments. However, there are some limitations in detection and mitigation of DDoS attacks, which might be considered a notable security flaw. Also, if an attacker becomes familiar with the principle, using which SecSip detects attacks, he can easily forge an attack to bypass this defense (using randomization in specific parts of the SIP messages).

3.4.4 Holistic VoIP IDS/IPS

The idea of a holistic approach to the problematics of detection and prevention attacks in the VoIP environment was proposed by Nassar et. al. [11]. This approach introduces two innovative ideas. First idea is using a honeypot (fake SIP environment) to attract the attackers, observe and analyze the attacks and use the information gathered from the attacks (attack sources and patterns) to detect and mitigate these attacks in a real SIP environment. The second idea points

out the fact that SIP environment has a distributive basis and therefore it might be useful to take into account information from more different sources when analyzing the SIP traffic (and consequently also attacks against SIP). The second idea lead to the implementation of a distributed event correlation mechanism that looks very promising for future research.

The paper states that a prototype implementation was tested in laboratory, however there is no publicly available implementation. The idea of a holistic approach was adopted also in a EUX2010sec project [12]. Practical results from this project should be available soon since its lifespan ends at the end of year 2010.

3.4.5 VoIP Defender

VoIP Defender was presented by Fiedler et. al. [13] and describes a framework of highly scalable SIP-based security architecture. VoIP defender uses modularity to achieve simple extensibility and scalability and introduces an idea of using information from all the layers (starting on the physical layer, all the way up to the application one). The proposed framework is quite complex, but the architecture offers adding and removing threat detection algorithms according to actual needs, which makes this solution very attractive.

However, as in the previous case, implementation of VoIP defender is not freely accessible and, as the paper states, only the skeleton of the framework is ready. The whole solution is therefore not ready for deployment in real SIP environments.

3.4.6 SIP-aware DDoS Attack Detection System

A design of a SIP-aware DDoS Attack Detection System was presented by Ha et. al. [14]. The principle of attack detection is based upon statistical methods and attack pattern recognition. Interesting idea is usage of a set of six logged information about each SIP packet combining the information from application layer with the information from the link layer.

Unfortunately the paper is not very detailed and there is no clue of where to find additional information or the implementation of the system (although the paper states that the prototype implementation should be available).

3.4.7 Conclusion

The mentioned projects are by no means an exhaustive list of projects related to my work. I have chosen only a few examples from the projects which I have

surveyed and there are surely other projects which I did not encounter in my survey. Interesting aspects which I have found in majority of the projects are

Absence of a complex attack simulation tool Each of the surveyed projects aims at analysis, detection or prevention of different attacks against SIP. Many projects contain description of these attacks and attack patterns, many states that the lab tests were done in lab environments but almost none describes which tool (tools) was used as an attack generator. I am aware that there are plenty of different tools that can be used for generating different types of attacks (concrete tools are mentioned later in the *Attack Realization* chapter), however, I have not found a single complex tool which is able to cover the majority of area of DoS and DDoS attacks on SIP. Specifically, I have not found a single tool that is able to produce a decent SIP DDoS flood attack.

Ignoring spoofing in DDoS flood attacks Some projects aim at DoS attacks but leave out the advanced version of DDoS attacks and the projects considering DDoS attacks often leave out the possibility of spoofed addresses. I suppose that the mechanism of source spoofing can be very useful in the field of testing of defense solutions deployed in a VoIP environment.

Absence of publicly available prototype Many of the projects aiming at defense against the attacks targeting SIP environments state, that lab tests were done using prototype implementations of solutions described theoretically in the papers, but these implementations are very often publicly unavailable. I understand that the prototype implementations are often programmed in a rush, with poor programming style and minimal to none optimization since the basis of the research is developing and designing new approaches and ideas to the topic being researched, however I suppose that it might be very useful to make these prototypes available for public to test and maybe someone will adopt and improve the solutions. This is especially crucial in case of projects which end with a report or conference paper and have no successor.

I have weighted these aspects and decided to try to fill in this gap in the rest of this thesis. Therefore I have focused on two main goals. The first goal is design and prototype implementation of an attack simulation tool which is able to produce both DoS and DDoS attacks of both malformed message and flooding attack types. The second goal is design and prototype implementation of a lightweight defense solution which is able to detect and mitigate these attacks (at least to some extent). I am focusing on the theoretical approach to the defense

solution, trying to bring some new ideas but I also provide you with a working prototype for your own tests and future research.

Chapter 4

Attack Realization

4.1 Introduction – background and reasoning

At first, I would like to denote that even though this part of the thesis focuses on realization of various SIP DoS and DDoS attacks it should by no means be considered something like a guide for hackers. I am aware that the tools and procedures described in this chapter can be misused for a real attack deployment, however, it is something that cannot be prevented. I am using the attack simulations to test the capabilities of SIP servers and deployed defense solutions and if I want to acquire usable results, my simulations must be as similar to real attacks as possible. If you plan to test the robustness and stability of your own deployed SIP environment, you are encouraged to use any of the described tools and processes. If you plan to misuse any of the described tools and processes for malicious purposes I should warn you that I have intentionally left out some crucial information that are not necessary in testing use but may be fatal in case of using these for a real attack deployment.

4.2 Preparation of an attack simulation

The first thing you need to do is preparation of a testbed. It is possible to make tests directly in your deployed SIP environment but I strongly discourage you from doing so. The results of the tests are often fatal for the SIP environment and even if you believe that your infrastructure is well protected and are running the tests only to prove it, it is a wise decision to create a copy of your real infrastructure either in an isolated network (if you have spare hardware you can use) or using virtualization.

Once you have your testbed ready, you should choose which type of attack you

want to simulate. If I use the classification which I proposed in chapter *Problem Analysis* then there are three main types of attacks to be tested (Well, four, but as I stated previously, the DDoS malformed message type is special and if you do the tests for both the DoS malformed message and DDoS flood types, it should sufficiently replace this one. Be warned that the implication in the opposite direction is not true e.g. using DDoS malformed message type does not cover the area of DoS malformed message and DDoS flood). You should be aware that the tests will never guarantee that your SIP environment is 100% safe since the coverage, especially for the malformed message type of attack, is far from being absolute. However, doing the tests can show you weak spots in your infrastructure and make sure that your environment is protected from the basic SIP-specific attacks.

Note that it is necessary to test all important SIP components contained in your infrastructure. This does not mean that you have to test every single SIP phone but you should definitely test every unique version of a SIP component which is accessible from the Internet. It might be also useful to do basic tests for the components which are normally inaccessible from the outside because if the attacker succeeds for example in compromising your border SIP server and this server sends a malformed message to all of your SIP components inside your private network, it would be a disaster if they all fail upon receiving this message.

4.3 Running the attack simulation

Once you have prepared your testbed and chosen which type of attack you want to simulate, you just need to find a tool suitable for generation of such an attack, configure it and deploy the attack. I could have written the list of tools along with prepared scripts for each attack type but that will lead exactly to the result I do not want to end with. Either these scripts will be used by script kiddies to produce some real attacks or some lazy administrators will use them in their SIP environments and state, that their infrastructure is safe because they have proven it by running the tests. My goal is to show the process of the attacks and point out, how the attack actually works and what weaknesses it tries to exploit, so that you understand the principle. I have already briefly described the basics of each attack type in the previous chapter so now I am going to describe the attack process for each of the mentioned attack types in more detail and provide you with a list of tools that can help you in preparing simulations of these attacks. In the next chapter, I present an attack simulation tool called SIPp-DD that I implemented by modification of the SIPp call generator [1] and which was designed to be able to produce all three mentioned types of attacks.

4.3.1 DoS malformed message attack

The malformed message attack aims to exploit the parser mechanism contained in each SIP component. Once the SIP message is received by a SIP component, it is processed by the parser which extracts the data from the message and provides them for additional processing. There are many SIP message types and the messages are not absolutely uniform (meaning that there are some optional parts which might or might not be present) and the parser must be complex enough to be able to parse all these well-formed messages. The problem is that the programmers often test only that the parser is able to process the well-formed messages before they release their SIP component and therefore if a malformed message is provided to the parser, it might crash the parser.

A specific type of malformed message is a message that does not contain a mandatory field. Even though many of the parsers of nowadays implementations are already able to handle such messages (or, better said, they do not handle them but drop them instead) there might still be some parsers unable to handle such message and processing of such message might lead to their crash. I mention this special case here because it might be easily forgotten in testing and that omittance might lead to crucial consequences.

There are two general approaches to simulation of this attack type. The first approach is to use a set of predefined malformed messages and the second is to use a "malforming engine", which modifies originally well-formed messages, which it gets on its input. Both approaches are quite common in security testing and have their strengths and weaknesses.

The first approach was very popular in the early phase of SIP spreading when there were plenty of prototype SIP stack implementations full of bugs. One of the most known tools that were developed in that time was PROTOS Test-Suite: c07-sip [15]. The tool is composed of two main components – a set of test cases (malformed messages) and a simple java application capable of sending the test cases to the target SIP component. Very similar to the PROTOS c07-sip tool was SiVuS, Sip Vulnerability Scanner. It worked on the same principle but had another set of malformed messages and was primarily designed for Windows OS. Even though I have used SiVuS in early stage of my research I have recently found out, that the webpage of the project was deleted and the domain is for sale. The same fate shared also SFTF, a tool working on the same principle, whose development was also abandoned and the webpage was deleted.

Even though the tools are no longer available, the basic principle is clear. Implementing an application capable of sending prepared SIP messages is a matter of days and the most important part remains the contents of the set of malformed messages. For an inspiration how to construct such set you can experiment yourself or see the already mentioned webpage of the PROTOS c07-

sip tool [15].

The advantage of the first approach is its easy implementation and completeness. If your SIP components handle all the malformed messages from the initial set without being negatively affected, then you can be sure that these malformed messages will not harm them in case they are a part of a real attack. However, the biggest disadvantage is the mentioned advantage taken from another point of view. You can be sure that your SIP component is capable of handling the messages in the initial set but no other messages are tested and so it might happen, that a malformed message which is only slightly different from any message in the initial set, will crash the SIP component. The question therefore is how big the initial set should be and how smartly chosen should be the malformed messages contained in it.

The second approach is based upon a well known security testing principle called fuzzing. Fuzzing is a testing method which uses randomly crafted inputs for an application in order to find the inputs which are fatal for the application and result in a crash, deadlock, inconsistent state etc. In case of SIP components this means providing the SIP component with randomly generated input and monitoring its state to detect failures. Because SIP messages have standardized structure and plenty of randomly generated inputs will be rejected right away since they will not begin with the standardized keyword or have the necessities whose presence is checked before the message is being parsed, it is wise not to use totally random input but instead use valid SIP messages that are "fuzzed" before being delivered to the tested SIP component.

For crafting of modified SIP messages you can either use one of publicly available fuzzing frameworks like Peach [16] (which will require manual specification of the structure of all SIP messages that will be crafted, using the tools meta-language) or you can use a tool called fuzzing proxy. Fuzzing proxy is basically a network proxy (e.g. tool sitting between A and B and delivering messages from A to B and in the opposite direction) which modifies the transferred messages (often pseudo-randomly). One implementation of such fuzzing proxy that is suitable for this purpose is HotFuzz [17].

The advantage of the second approach is the increasing percentage of coverage. The longer the simulation runs, the more messages are malformed and processed and therefore the number of malformed messages that the SIP component is able to handle increases. Another advantage is that, thanks to the pseudo-random modifications, one can find a pattern of dangerous SIP messages capable of crashing the SIP component which will be very hard to identify otherwise. The disadvantage of this approach is the fact that there is no guaranteed coverage and the simulation might take quite a long time (since it can generally run infinitely).

A good idea might be to combine both approaches. Manually craft a set of malformed SIP messages based upon potential weaknesses derived from the survey of the SIP standard and after testing these, use the second approach and let the simulation run for a couple of days or weeks (according to available resources). The SIPp-DD tool which is presented in the latter part of this thesis is capable of both approaches.

4.3.2 DoS flood attack

The DoS flood attack aims at exhausting the resources of the target SIP component. The most often targeted resources are CPU usage and memory. The principle of the flood attack is quite simple – the attacker generates a huge amount of SIP messages which are being sent periodically to the target SIP component. The SIP component tries to handle these messages but because there are so many of them, the component soon runs out of resources and is unable to process legitimate SIP messages. Which resource is exhausted at first depends upon the implementation of the SIP component as well as upon the choice of messages that the attacker uses in his attack.

Creating a SIP DoS flood attack is very easy, one just needs a generator of SIP messages (the tools providing this functionality are often called call generators). It is not necessary to implement a new generator since there are many of them publicly available. One of the most known and used free ones is SIPp [1] which has a good command line interface and is able to generate any part of SIP communication. It also offers additional services like creation of the RTP stream with real voice data etc. but these functionalities are not important in scope of this attack simulation. Another good free call generator is SIPr [18].

Important for this type of attack is the choice of SIP message type (or combination of message types) that will be sent to the target SIP component during the attack. I have done some research in this field to find out which message types are the most resource consuming ones when being processed. The results show that there are three types of SIP messages especially suitable for flooding and these are INVITE, REGISTER and OPTIONS. Processing of these three SIP message types consumes the most of resources of the targeted SIP component. Surprisingly, there is no notable difference in resource consumption whether you use a uniform attack composed of messages of only one of these types or composed of a combination of these three message types. Therefore it easier to use the uniform version of the attack.

The advantage of this attack is its very easy deployment. The call generators are rather complicated because they offer plenty of additional functionalities but if one needs only the basic functionality of sending a uniform flow of SIP

messages of the same type then it can be done using a one line command for the SIPp.

The disadvantage of this attack is, as I have already showed in the previous chapter, existence of trivial, yet very effective, countermeasures, that are able to mitigate the attack completely.

One important fact concerning this attack, that has not been mentioned yet, is whether to use any secret information if the attacker has access to them. By this information I mean especially valid usernames or call numbers registered at the target SIP server (in case of SIP client, this is meaningless). Up till now I have been considering the generated requests to use generated (invalid) information about the caller, callee etc. Will it be effective to create SIP requests with valid information instead – for example to use an INVITE request with valid callee? The answer is a bit surprising. Even if the attacker has valid information about a registered username (number) it is not better to use these in this type of attack since it will not increase the impact of the attack on the target SIP server. The reason is obvious, if you give it some thought. If the information about the callee is invalid, the server has to go through the whole list of registered users to check, whether such a user exists. If the information is valid, however, the server does not have to go through the whole user database and therefore the impact of the attack might be even decreased. The situation is a bit different for the REGISTER request, if you use a valid username.

Using REGISTER flooding with valid usernames in a DoS flood attack might lead to a very specific resource exhaustion – the server might run out of nonces. To understand this attack you must first understand the process of registration. The SIP registration process looks as follows

1. Client sends a REGISTER request to the server (without digest)
2. Server responds with a 401 Unauthorized response containing information necessary for registration (nonce, realm and digest algorithm)
3. Client computes its authentication digest and sends new REGISTER request with this valid digest
4. Server sends 200 OK and inserts the location information (IP address and port) into the database

For security reasons, the nonce should be unique for each parallel registration process. That means that the server generates the nonce for each registration process that began with a REGISTER message and keeps it until the registration process successfully ends or until a lifespan of the nonce ends. The number of nonces that can coexist in the same time as well as the lifespan of a nonce

depends on the implementation of the SIP server. I was able to successfully reproduce this nonce exhausting attack against current version of one very popular implementation of SIP server during my testing, proving that even nowadays servers are prone to this attack.

4.3.3 DDoS flood attack

The principle of the DDoS flood attack is the same as for the DoS attack. The only difference is that a DDoS flood attack has many sources meanwhile the DoS flood has only one. However, this only difference is of utmost importance because it means that the situation for both an attacker and defender changes considerably. I have already briefly mentioned the impact of this difference on the defense mechanisms in the previous chapter and will discuss it more thoroughly in the chapter describing the SIPp-DD testing tool. Now I focus on the impact of this change on the attacker.

For producing a SIP DoS flood, the attacker nees one average computer connected to the Internet and a call generator – almost anyone has access to a computer connected to the Internet nowadays and there are some freely available call generators on the Internet, so these requirements are easily fulfilled. Producing a DoS flood against any SIP component accessible from the Internet is therefore very easy even for an average computer user.

The requirements necessary to produce a DDoS flood differ upon the approach you choose. There are two general approaches to DDoS flood generation – one can use many different real sources or use only one source, which pretends to be many different sources. Both approaches have their advantages and disadvantages.

The first approach is generally simpler but requires, that the attacker controls a lot of computers. These computers must additionally not be in the same private network (because if they are, the NAT on the border of the network will usually map them to the same public address, thus diminishing the distributiveness of the attack). The attackers often fulfill this requirement by using a botnet (an attacker can acquire a botnet from various sources – using a worm, buying on the black market. . .). Once the attacker controls a botnet, producing a DDoS flood is not hard at all since it is only necessary to start a call generator on each of the bots (like it was done in case of a DoS flood on the attackers machine). Installing and starting the call generator on each of the bots will be very boring and time consuming labor, however even that can be solved using distributed scripts. If you do not know what these distributed scripts are or how to use them, it only means that you are not an attacker and in that case, the second approach will be surely better for you (since it was designed precisely for testing purposes).

The second approach uses the technique called spoofing. I have already mentioned this technique in the previous chapter. Spoofing is a method to inject values of source IP address and port into the packet to be sent and therefore the receiver of the packet will suppose that it came from someone else than it actually had. Spoofing has its limitations – NAT traversal, defense mechanisms based upon IP traceback, egress filtering... However, if you use the spoofing mechanism for testing purposes inside a private network, then none of these limitations affect you in any way. Very important is also the fact that spoofing, being relatively simple to implement for UDP transport, is quite hard to implement for TCP transport and almost impossible to implement for TLS transport. Thankfully, I do not mind that since, as I have already mentioned in the beginning of the thesis, I focus on SIP transported over UDP only.

When I was looking for a SIP testing tool using the second approach, I have found none. Of course one can use some of the low-level network libraries in various programming languages and implement such a tool, but it is not a trivial work and one has to have a good understanding of network principles and programming techniques. Therefore I have decided to write a testing tool with the spoofing functionality, because it can definitely be useful when testing resistance of SIP infrastructures to DDoS attacks. I have named my tool SIPp-DD, which is an acronym for SIPp-Distributed DoS, and details about it as well as description of the implementation of the spoofing mechanism are available in the next chapter.

Chapter 5

SIPp-DD: A complex SIP DoS and DDoS attack simulation tool

5.1 Motivation and initial requirements

As I have already shown in the previous sections of this chapter, there are plenty of tools that can be used for attack simulations targeting SIP infrastructures and testing its defenses. However, none of the tools which I have found and tried was capable of providing all the functionality which I needed for my tests. The tools designed for testing SIP components with malformed messages are usually not very flexible, the call generators offer plenty of functionality useful for fully-fledged SIP traffic simulation but are not designed for manipulation with malformed messages and the flexible SIP testing tools (like sipsak [19]) have poor or no traffic rate control. The biggest problem, however, is the absence of support for simulation of distributed attacks. None of the tools I have tested has support for distributive synchronization or IP spoofing. Because of these facts, I have decided to design and implement my own tool with all the functionalities I find important for testing of DoS and DDoS attacks against SIP environments.

To summarize and keep in mind all the necessary requirements, I have prepared the following list with the initial requirements:

- Support for distributed attacks
- Support for easy SIP message modification
- Traffic rate control and time control
- Support for injection of external values
- CLI for easy and structured control

- Script support for creation of advanced scenarios

5.2 Critical decisions

The first critical decision was whether to build the tool from scratch or to modify one of the existing tools and add the wanted functionality. As is obvious from the name of the tool, I have chosen the second way and modified SIPp [1]. There were a few reasons for this decision. The first and most important reason was that building a tool from scratch would require much more time and resources than modification of an existing one. The second reason was that when I was using SIPp in my tests, I became familiar with it and I liked the way it is controlled. The third very important fact was that SIPp is open source and it is therefore no problem to modify the existing source code or extend it in any way you need. Since SIPp already fulfilled lots of initial requirements I have stated for my tool, there was no additional deciding necessary and I have adopted it as a basis for the new tool.

The second critical decision was how to implement the missing necessary functionalities. After I have thoroughly inspected SIPp, I found out, that the biggest challenge will be implementation of the support for distributed attacks. All other requirements were already fulfilled or seemed to require only slight modifications of the current SIPp implementation. There were two options from which I could choose. Either create a framework for remote control and synchronization of more running instances of SIPp or integrate a spoofing mechanism into SIPp. Considering the fact that the first option would require many computers to simulate a DDoS attack and that I need only the support for UDP, I have chosen to implement the spoofing mechanism into SIPp.

5.3 Design

I have adopted the whole basic design of the SIPp because it is from my point of view very convenient for the desired purpose. Since the design is only retaken from SIPp, I will not go into details (these are available on the web page of SIPp [1]) and I will only briefly describe the overall structure with the most important aspects.

SIPp-DD is controlled from the command line the same way original SIPp is. I am using an XML script to define each attack scenario. The XML script contains the concrete SIP messages to be sent during the scenario. Next to these messages, there are also possibilities to set various pauses between the messages and state various conditions inside the XML. Very important is the possibility

to insert values from external CSV files into any part of the SIP messages, that will be used in the scenario. This way you can easily specify a CSV file with information that will be inserted into the SIP messages during their creation in a simulated attack. Moreover, SIPp has already implemented two different access politics to these files meaning that the values can be read either sequentially or randomly from the external CSV file. This way it is very straightforward to implement a malformed message attack – you just need to create a CSV file with the modifications you want to do in the generated SIP messages and then set appropriate part of the generated SIP messages inside the XML file to be read from the external CSV file and that is it. Address of the target SIP component, the attack rate and various other options can be specified on the command line.

I am aware that the previous description may be a little compressed, so maybe the things will be more clear if you have a look at Fig. 5.1

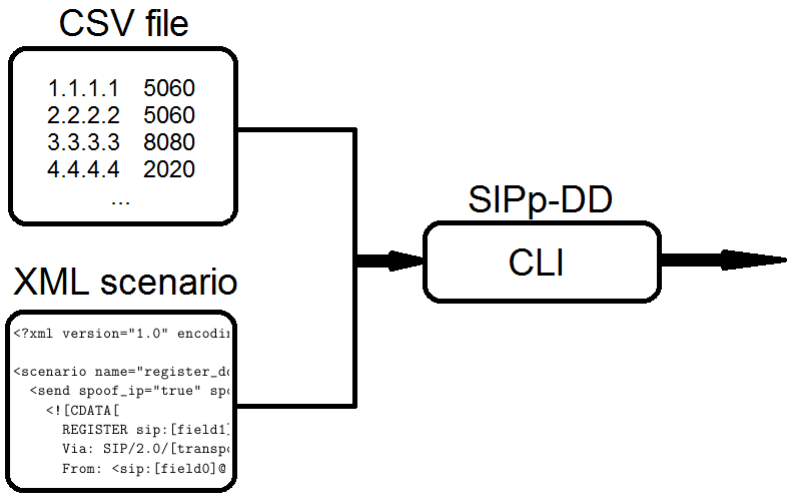

Figure 5.1: Architecture of the test deployment using SIPp-DD.

The design is very straightforward and clear, if you use the tool and so I will try to demonstrate it also on a practical example later.

5.4 Implementation

SIPp-DD was created by direct modification of the SIPp source code, version 3.1-TLS. All the sources are written in C++. I have prepared two distributions, both are enclosed on the CD. The first is the complete gzipped source code along with some additional examples and can be found in *prototypes/SIPp-DD/sipp_dd.tar.gz*. It is enough to unpack the archive and use the standard

make. The second distribution is only the patch applicable to the original SIPp source code. I have created the patch so that it can be easily updatable to the new versions of SIPp. It can be found in *prototypes/SIPp-DD/sipp_dd.patch*. To apply the patch, use the classic unix *patch* command.

Note that SIPp is primarily a unix tool and its functionality on other systems is not guaranteed (so the same goes for the SIPp-DD). Also, for proper functionality of the spoofing mechanism, you have to allow the nonlocal bind functionality in your operating system. This is usually done by setting the *ip_nonlocal_bind* variable to true (respectively 1). This variable can be found in the IP protocol implementation, usually in */proc/sys/net/ipv4*.

As for the details concerning the changes in the SIPp source code, you can find them in the already mentioned patch file *prototypes/SIPp-DD/sipp_dd.patch*. The source code of SIPp is not very well commented but it has a good structure – the name of the source code file corresponds with its contents. . . mostly. Therefore I tried to stick with this structure and tried to insert my modifications and new parts of code to the corresponding files.

Most importantly, I had to add support for the usage of RAW socket so that the IP spoofing mechanism can be used. This required modification of a couple of different files.

At first, I had to change the sending mechanism. The existing implementation of SIP message sending mechanism in SIPp used a *write_socket* function with flags specifying the type of socket to use. This approach was not suitable for me, because I needed to provide the RAW socket with other data than the classic UDP and TCP sockets expect (especially with the values of IP and port to be spoofed). I had to add an additional method of sending packets using the RAW socket, which I named *send_message_raw*. I have inserted this method directly in *sipp.cpp*. To be able to create a packet with the spoofed address and port, I also had to add a function for computation of checksum for the headers and functions for creation of wrappers around the RAW socket, so that SIPp registers its existence and can work with it. The corresponding functions are also implemented directly in *sipp.cpp*.

I also had to introduce new options that can be inserted in the scenario XML files specifying whether to use the IP spoofing mechanism and which columns of the presented CSV files contain IP addresses and/or ports. This required modification of *scenario.cpp*, where I added some new static variables to store the necessary values and handling functions for these variables.

I also had to slightly modify the way SIPp handles the reading from the CSV files. This modification was done in *call.cpp*. In the same file, *call.cpp*, is also the place where the *write_socket* function was called in the original SIPp. This place was changed such that either the original *write_socket* function can be called,

if the spoofing mechanism is turned off, or the necessary information are loaded from the external CSV file and the *send_message_raw* function is used, if the spoofing mechanism is active.

Some changes were also done in the header files *sipp.hpp*, *scenario.hpp* and *call.hpp*. These changes were only definitions of variables and functions to maintain the structure of the declaration-implementation structure. Generally, I tried to make as few changes as possible.

The usage of SIPp-DD is very similar to the usage of SIPp, because it inherited many of its functionalities. A good example of usage of SIPp-DD can be found in the following section. I have focused on the implementation of functionality rather than taking care of side effects so it might happen that the program crashes if you use a bad configuration (such as combination of TLS or TCP transport mode with active source spoofing mechanism). I am aware of these shortages, however I did not aim to create a full-pledged application and I focused on further research rather than on fine-tuning of the tool.

5.5 Example of usage

In this practical example I want to demonstrate the usage of SIPp-DD for a simulation of a DDoS flood attack. In this attack I use the REGISTER request with crafted (invalid) credentials. The attack is uniform and I use the rate of 500 requests per second. The attack ends after sending 50000 messages (which means the attack duration is 100 seconds). The hostname of the attackers machine in this example is "malicious", real IP address is 192.168.6.25. The hostname of the victim server is "innocent", port on which the SIP server runs is typical SIP 5060 and the IP address is 192.168.66.15.

At first I need to create an XML scenario file named "test.xml". In this example file I use only the most important necessities, a real scenario might be much more sophisticated. Therefore I need to specify only the action "send the request", the full body of the SIP message that will be sent and options specifying that I want to spoof the IP addresses of the messages being sent.

Contents of the *test.xml* file:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<scenario name="register_ddos">
  <send spoof_ip="true" spoof_ip_field="0">
    <![CDATA[
      REGISTER sip:[field1] SIP/2.0
```

```
      Via: SIP/2.0/[transport] [local_ip]:[local_port];
        branch=[branch]
      From: <sip:[field0]@[field1]>;tag=[call_number]
      To: <sip:[field0]@[field1]>
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Contact: sip:[field0]_[branch]@[local_ip]:[local_port]
      Max-Forwards: 5
      Expires: 3600
      User-Agent: SIPp/Linux
      Content-Length: 0
    ]]>
  </send>
</scenario>
```

The *spoof_ip* option sets that I want to spoof the IP address of the requests being sent, the *spoof_ip_field* specifies the column in which the IP address is in the CSV file. So the next thing I need is the CSV file which has IP addresses in the zeroth column. I am using the "addresses.csv" file.

Contents of the *addresses.csv* file:

```
SEQUENTIAL
10.10.10.1
10.10.10.2
10.10.10.3
10.10.10.4
10.10.10.5
10.10.10.6
10.10.10.7
...
10.10.10.255
```

The *SEQUENTIAL* statement at the beginning of the file specifies the access method that SIPp-DD will use when reading this file. The list of addresses which will be used as spoofed sources in the attack follows.

Finally I need to start the generator with appropriate options. The starting command I use is:

```
./sipp -sn uac innocent:5060 -sf test.xml -inf addresses.csv
 -m 50000 -r 500
```

SIPp-DD is started using the *./sipp* command. The *-sn uac* option specifies that I am emulating a client, *innocent:5060* is the identification of the target, *-sf test.xml* specifies the source XML file with the scenario, *-inf addresses.csv* specifies the CSV file with the list of IP addresses to be used, *-m 50000* specifies the total number of SIP messages to be sent and *-r 500* specifies the number of SIP messages to be sent per one second. And that is all. After using the previously described command to start the SIPp-DD, the test will automatically generate the 50000 SIP messages that will be sent to the *innocent* machine with the rate of 500 messages per second.

5.6 IP address generation

SIPp-DD was designed with easy support for injection of external values. Therefore it is no problem to use an external CSV file containing the addresses to be used as spoofed source addresses in a simulation of a DDoS attack (as I demonstrated in the example of usage). The question that remains is: Where to get the file with the IP addresses to be used? Writing the file manually will be very ineffective so I decided to create a simple bash script for this purpose. Afterwards, I have decided to improve the method of IP address generation to simulate more real-like attacks and developed an advanced version of the generator using Matlab.

5.6.1 Simple generator

The simple generator is a very straightforward bash script. When starting the script you need only to specify two parameters – how many addresses you want to generate and the name of the file that should be created. The process of generation is then fully automatic and the addresses are totally random. Usage of the script follows.

```
./ipgen_simple 8 addresses.csv
```

Contents of the generated *addresses.csv* file:

```
SEQUENTIAL
217.189.177.48
245.51.209.48
100.128.134.234
112.189.250.59
53.221.30.162
```

```
9.163.92.125
247.143.237.22
90.16.35.51
```

The script automatically inserts SEQUENTIAL keyword at the beginning of the file to indicate that the IP addresses should be read sequentially from the file, when used. Since the addresses are already generated randomly there is no need to use the RANDOM method. The implementation of the script is enclosed on the CD in *prototypes/generators/ipgen_simple*.

Even though this script is usable for some basic tests, it also has some limitation. Thanks to the absolute randomness, next to commonly used IP addresses, it produces also non-routable IP addresses, IP addresses from unassigned blocks etc. Since I am aiming at producing real-like simulations, this generator is not good enough.

5.6.2 Advanced IP address generator

I have faced two main challenges when developing the advanced IP address generator. Split into separate questions, these challenges were

- Are there any IP addresses (or blocks of IP addresses) that are being used more often in attacks? If so, which addresses they are?
- Are there any typical patterns of a DDoS attack process?

Identifying the distribution of "maliciousness" in the IP address space

When looking for clues to answer the first question, I have found a Hilbert map of malicious activity which can be seen in Fig. 5.2.

Even though Team Cymru considers the background information needed for creation of the Hilbert map their internal knowledge which they do not share, they have given me permission to re-use the picture of the Hilbert map and extract the information I need directly from the picture.

Because I have not found a better source of the desired information concerning the attack load between the concrete blocks of IP addresses, I decided to do an analysis of the Hilbert map. At first, I had to choose the granularity of the information I wanted to extract. Because one pixel represents a block of 4096 IP addresses, I could not get under this limit. Therefore I have decided to extract the information about only the first two bytes of the IP address and generate the last two bytes randomly.

The extraction process of the first byte was done as follows: I unwrapped the Hilbert line from the square to obtain a straight line of smaller squares, each

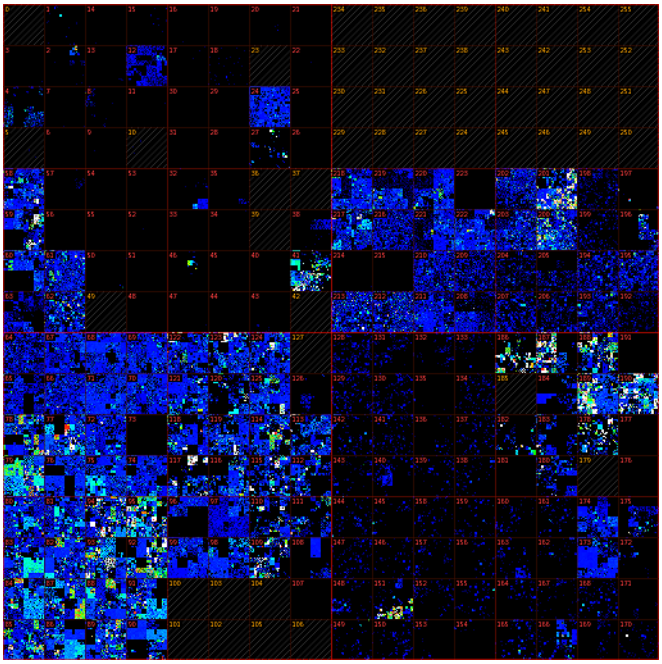

Figure 5.2: Hilbert map of Internet malicious activity created by Team Cymru, 2010 [20]. The map shows the entire Internet address space, each pixel representing a block of 4096 IP addresses. Pixel color represents level of malicious activity produced by machines with IP addresses from the corresponding block (heatmap scheme: black = none, white = highest).

containing pixels corresponding to one /8 subnetwork. The squares were then sorted in the line from 0 to 255 so I just had to count the number of differently colored pixels in each square. As the colors were taken from the whole color palette, I assigned every individual pixel a weight computed according to its color (eg. black=0, blue=1, purple=2, green=3, yellow=4, orange=5, red=6 and white=7). The final weight, computed as a sum of all pixels weighted colors in the square was assigned as a weight to each of the 256 squares. Finally the weights were normalized over the sum of all weights to derive the probability for the distribution function. The resulting cumulative distribution function can be seen in Fig. 5.3.

To extract the necessary information about the second byte for each of the 256 first bytes, I had to divide the smaller squares forming the previously unwrapped Hilbert line into 256 parts. As every square was formed by 63x63 pixels (62x62 for the squares at the map edge), I have used interpolation from the nearest pixels to create a borderline to complete the desired 64x64 square. Each square was then converted to a 16x16 square (every 16 pixels forming a small square

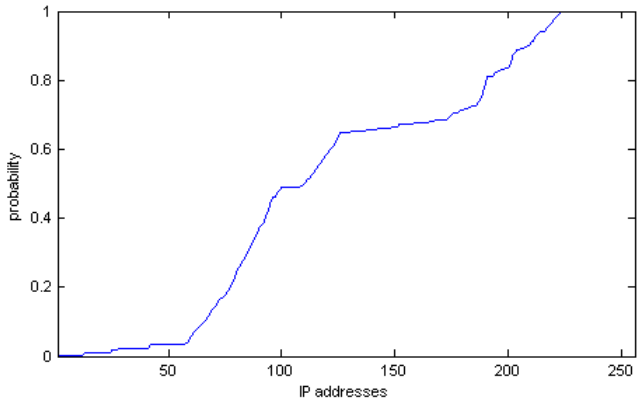

Figure 5.3: Cumulative distribution function of the aggregate level of malicious activity computed for /8 network blocks.

were aggregated into one with weight equal to the sum of its weights). Because every resulting square again represented 256 bytes ordered in a Hilbert curve style, it was necessary to unwrap each to obtain a straight line from 0 to 255. Finally the weights were normalized as in the previous part.

The results of the analysis are distribution functions which I obtained for the first byte of the IP address and consequently for each of the second bytes of the IP address. When the advanced generator produces an IP address, it firstly pseudo-randomly chooses the first byte (using the distribution function computed for the first byte) and then pseudo-randomly chooses the second byte (using the corresponding distribution function computed for the already generated first byte). The last two bytes are generated pseudo-randomly using a discrete uniform distribution.

Identifying typical patterns of DDoS attacks

To identify typical patterns of DDoS attacks, I needed some traces of such attacks. I have obtained a few traces from the USC/Lander project [21] and also some aggregated data about one DDoS flood attack against an HTTP server [22]. Even though the dataset was quite small (I have identified only four different DDoS attacks in the traces), I was able to extract some patterns that might be common for DDoS attacks.

For the purpose of the analysis, I have marked the attacks with letters from A to D. For the first three attacks (A–C), I had anonymized traces containing the timeline of the attack, for the fourth one I had unanonymized aggregated data about the number of packets from different sources involved in the attack.

The aggregated numbers computed for each attack are available in Table 5.1.

Table 5.1: Data gathered in analysis of the DDoS traces.

	number of packets	number of IP addresses	duration
Attack A	1043222	145	251s
Attack B	22603325	3	816s
Attack C	195021	28	40s
Attack D	749778645	4217	unknown

First, I focused on the time process of the attacks. Observing the attacks A–C (for which I had the timeline), I found out that the attacks began almost simultaneously (all the involved attack sources started their cooperative attack in a few seconds) and ended the same way. This behaviour is logical since the attacker needs to flood the server for which he needs to accumulate the attack load and there is no point in starting or ending the attack slowly. Instead, it seems wise to concentrate all the traffic into the same time period. If I suppose that the attacker has used a botnet for the generation of the attack load and that the botnet was controlled using a central management script, then centralized points of both attack start and attack end makes sense. Therefore I have taken this synchronized behaviour as the first pattern for the DDoS attacks I want to simulate.

The second important pattern seemed to be the share of the attack load between the individual sources. The attack load in attacks A and C is shared among distinctive groups of sources with similar attack rates. This can be clearly seen in Fig. 5.4.

For attack B, there are only three sources and much more packets involved. Therefore I did not plotted the numbers of attack B into the graph because it will make the graph unpredicative. However, the attack load is shared among the individual sources such that the first source had generated approximately 95% of all the packets involved in the attack and the remaining two sources generated 2.64% and 2.34% respectively. This share distribution complies with the idea of isolated groups with similar attack rate derived form attacks A and C. As for attack D, one can also find a few distinctively isolated groups which can be seen in Fig. 5.5.

Even though the trend of groups with similar attack rate is not so clear in attack D, it is still present and so I take it as the second pattern for my simulator of DDoS attacks. It might be useful to analyze more traces in the future work, preferably of SIP DDoS attacks, if I will be able to gather any such traces.

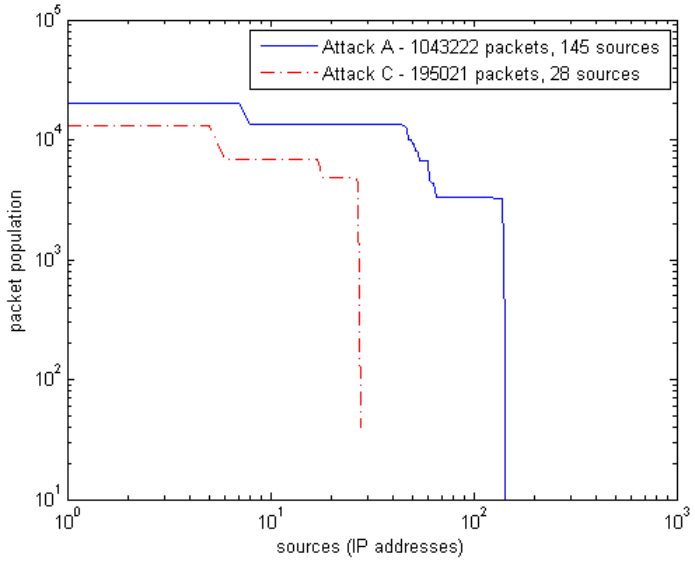

Figure 5.4: Loglog scale of DDoS attack load distribution between the attackers for two ICMP flood attacks.

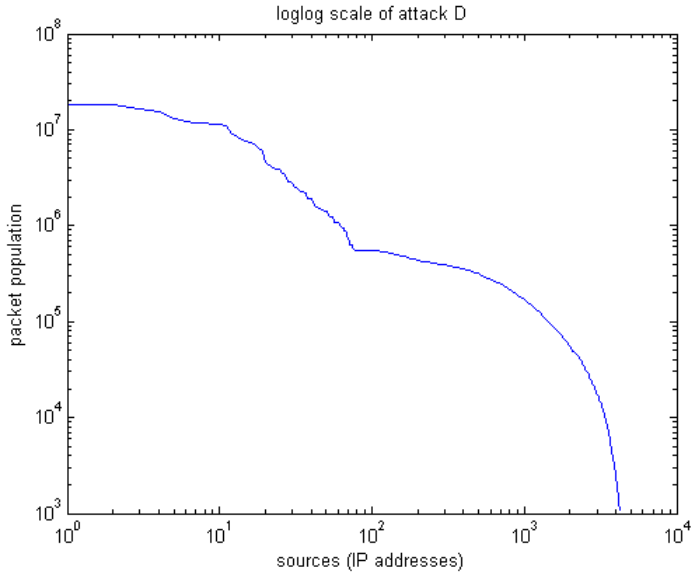

Figure 5.5: Loglog scale of DDoS attack load distribution between the attackers for attack D.

Implementation

I have taken the two patterns typical for DDoS attacks which I identified in the analysis and implemented an advanced IP address generator as a script for

Matlab. Because SIPp-DD expects a list of IP addresses in a form of a CSV file, the script produces such a file. The IP addresses are generated randomly using the distributions derived from the Hilbert map analysis and therefore no non-routable or non-assigned addresses can be present in the generated set of addresses. Moreover, the resulting distribution in the set of IP addresses generated by the script roughly complies with the distribution derived from the analysis of the Hilbert map (the grouping mechanism of course dislocates the distribution, still, the correlation coefficient of the distribution in the generated sets with the distribution derived from the Hilbert map is usually higher than 0.3). The pattern of grouping with similar attack rate was achieved by repeating the generated IP addresses in the resulting CSV file. The implementation of the grouping mechanism showed to be somewhat complicated. Because I needed the generator to be suited especially for the SIP DDoS attacks which have an attack rate between a few hundred and a few thousand messages per second, I optimized it for these values (but I am aware that a more flexible solution should really be done in the future work).

The inputs for the generator are number of IP addresses to be generated and number of messages to be generated in the whole attack. The final mechanism of the generator then works as follows:

- The total number of IP addresses to be generated (equals the attack rate in messages per second) is divided into four groups. The first group contains 30% of IP addresses, the second 50% of IP addresses, the third 20% of IP addresses minus r (where r is a random integer between 1 and 5, but never higher than 10% of IP addresses) and the fourth r addresses.

- The whole load of the attack (the total number of messages to be sent during the attack) is divided into four groups. The first group takes 40% of the load, the second takes 50%, the third takes 9.9% and fourth takes 0,1%

- The concrete IP addresses are generated randomly using the distributions derived from the Hilbert map analysis for the first two bytes and a discrete uniform distribution for the last two bytes for each address.

The main file, containing the implementation of the described IP address generator, is enclosed on the CD in *prototypes/generators/IPgen.m*. The necessary information about the distributions of the firts two bytes of generated IP addresses are stored in *prototypes/generators/ipdistro.mat*.

Example usage

I tried to automatize the process as much as possible and so the usage is very simple. After opening Matlab, you need to import data from the provided "ipdistro.mat" file. This will load the information about the distributions of first two bytes of IP addresses derived from the Hilbert map during the analysis process. These information are necessary for the generator to work properly (even though you might change these distributions if you have your own or you want to increase the presence of any particular blocks of IP addresses in the resulting file). Then you have to copy the file "IPgen.m" into your Matlab directory (or add its current location on the Matlab execution path) and run the script as follows:

```
distribution = IPgen(ffirst, ssecond, messages, sources)
```

The *ffirst* and *ssecond* variables contain the imported distributions for the first two bytes of the generated IP addresses. The *messages* variable represents the total number of messages to be send in one second by SIPp-DD (in scope of the generator script it refers to the total number of generated IP addresses in the resulting file). The *sources* variable represents the number of different IP addresses to be contained in the resulting file (unique sources). After the script ends, the *distribution* variable contains the distribution of the attack load among the generated IP addresses and the generated file "ips.txt" contains the actual set of generated IP addresses. This file can be directly used in SIPp-DD as the source CSV file containing the IP addresses to be used as spoofed sources in an attack simulation.

Chapter 6

New Defense Solutions

6.1 Introduction – Background and Reasoning

Many projects concerning security in the field of SIP infrastructure were done and many are in progress nowadays. SIP developed towards being more secure by introducing the TLS transport mode for signalization and SRTP (ZRTP) for media transfer. Many defense solutions were presented on various scientific conferences. And still, many of real and practically used SIP deployments are extremely vulnerable. There is not much I could do to solve this problem. It is a fact that deployment of a SIP environment is much easier (and cheaper) when focusing on functionality and not on security (that is the reason why the UDP transport mode is the most used one and some softphones and even hardphones are not able to practically use the TCP nor TLS transport modes). I am therefore trying to point out at least the biggest weaknesses of typical SIP environments and the easiness of disrupting the SIP service using various types of attacks (as I have shown in the previous parts of this thesis). I also try to present some ideas concerning the defense against many attacks targeting SIP infrastructures and show that their practical implementation is not that hard as it might look like.

My goal is not to present just another theoretical all-covering defense approach whose implementation will be unmanageably difficult. I am aware that it is impossible to create a 100% reliable defense solution and I do not aim that high. Therefore I decided to focus on attacks using one specific mechanism – source spoofing. I have already shown that using the source spoofing mechanism, one can easily create a SIP DDoS flood attack. Additionally, this mechanism can be even more dangerous if the targeted SIP environment uses a simple defense based upon limit thresholding combined with source IP filtering. In this chapter, I want to demonstrate the problems connected to the mentioned mechanism and also an innovative way of defense against attacks using this mechanism.

I will also present a design of a more complex defense solution aiming at mitigation of the most dangerous DoS and DDoS attacks targeting SIP environments. The focus is placed on the flexibility and modularity of the solution so that it may be used as a basis for implementation of different defense solutions in future work. Note that I focus primarily on defense mechanisms against flooding attacks. As for the malformed message attacks, I have mentioned some approaches how to defend against these in the chapter *Attack Realization* and I plan to focus on them in the future work (I provide some hints where the extensions can be done in the proposed defense solutions to cover these attacks too).

6.2 The Aspects of the Source Spoofing Mechanism

I have already mentioned the source spoofing mechanism few times in this thesis but I never got into details. However, for a good understanding of the following sections, it is necessary to understand the spoofing mechanism well. To understand the source spoofing, you have to understand the basics of the IP protocol at first. Supposedly you already know these, but I will briefly summarize the most important facts. IP headers contain, among others, information about source IP address, source port, destination IP address and destination port. IP protocol has a trusting nature e.g. there is no authentication mechanism for the information about ports and addresses contained in the IP header. Therefore if I capture an IP packet and replace the information in the IP header, noone will notice (if I do not forget to recompute the right header checksum). Moreover, I can decide not to use the standard and commonly used API for network communication (TCP and UDP sockets) and use a RAW socket instead. RAW socket offers the possibility to create the protocol headers for the packet to be sent manually, so one can insert any values he wants to any columns in the headers. This way, I can easily craft a packet with any source IP address and port I want and send it to its target.

The receiver of the packet with spoofed source information processes its payload and if he wants to react, he will send his reaction to the address which was spoofed e.g. to someone else than to the one who actually sent the packet. This behaviour is very dangerous and can be exploited in numerous ways. I have to admit that source spoofing is not generally that easily produced as I have presented it. It is no problem to spoof the sources in packets that are being sent inside a private network, however if you want to send packets with spoofed sources into the Internet, there are some complications. NAT traversal, border

control (egress and ingress filtering), router policies and many other obstacles might stand in the way. Also there is a very popular technique of IP traceback evolving rapidly in the past few years, that can be used to identify packets with spoofed sources. Still, getting a packet with spoofed source through the Internet to the desired target is doable with some effort and the results might be gruesome.

In this case, I am using this mechanism in a SIP DDoS attack simulation so that the packets involved in the attack seem to be from different sources even though they were sent from a single machine. It is obvious that using source spoofing can easily bypass the defense mechanisms based upon thresholding and source IP filtering. The basic principle of these mechanisms is quite easy – a threshold (limit value) is specified for a concrete type of traffic (such limiting rule may state for example that only 50 packets may arrive at the port 5060 from a single source IP address per one minute). If the threshold is reached, the corresponding source IP address is blocked from sending any more packets to the target server for a predefined period of time. Therefore if I use the source spoofing mechanism, generate random different IP addresses and use them as source IP addresses in the packets that I send to the target, the rule will be never triggered. Moreover, I have advertised that there exists a possibility to exploit simple defense mechanisms based upon thresholding and source IP filtering... so how can that happen?

Imagine a following situation: Target server uses defense mechanism based upon thresholding and source IP filtering. The thresholding rule on the target server is configured such that only 10 packets can arrive at port 5060 from a single IP per minute. Once the threshold is reached, the IP address that triggered the rule is blocked using a firewall rule for 10 minutes. The attacker knows that some clients of the target server have addresses in the subnet 192.168.15.0/24. Therefore he designs his attacks such that 10 packets are send every second and as a set of spoofed addresses he uses a sequence 192.168.15.0... 192.168.15.255. What is the result? All clients from the subnet 192.168.15.0/24 are blocked on the firewall of the targeted server even though they actually did not violate the rules. Moreover, if there was any additional rule against a general flood attack not sensitive to the source IP addresses, this rule will not be triggered thanks to the relatively low attack rate.

From the example above it is obvious that the threat, that can be produced using the source spoofing mechanism, is serious and should be definitely taken into account when designing the defense solutions. What might happen if the attacker in the example situation used the addresses of the real network interfaces owned by the target server? What if he used the address of the loopback interface?

6.3 Defense Against Attacks Using the Source Spoofing Mechanism

6.3.1 General Idea

I have already mentioned the method of IP traceback. It was designed specifically to identify packets with spoofed sources. However, even though there are some different implementations of this method, they often require a relatively high number of packets for their decision. Therefore they are not very suitable to be used in a SIP environment since the SIP traffic is generally not very high (it might be interesting to do some experiments to prove this assumption). I decided to try another way.

The biggest advantage of the spoofing mechanism is the fact that the real originator of the packet is masked. This way, the receiver of the packet cannot contact the real originator and if he wants to respond to the packet, he will try to contact the machine with the IP address that was spoofed in the packet. However, this behaviour is a double edged sword. If I am able to create a situation where the response from the target is necessary for the actual attack deployment, then the attacker using the spoofing mechanism will be unable to proceed with the attack because he will never get the response. Thankfully, in the SIP environment, there is a simple way how to achieve such situation – usage of the SIP redirection mechanism.

6.3.2 Architecture

Instead of having one SIP server on a publicly known address which is contacted by clients, you need two SIP servers. One real, fully functional SIP server that is used to handle the actual SIP requests and one SIP redirect server. SIP redirect server is a very lightweight version of a SIP server with only one important functionality – respond to every SIP request using a 30X SIP redirection message (where X stands for a number specifying the actual subtype of the redirection response). To achieve the goal, the location of the redirect server is published and the location of the fully functional SIP server is kept private (it is contained only in the SIP redirection messages generated by the redirect server). Preferably, the two servers should run on different machines (though you might use virtualization if you lack the hardware resources).

The actual SIP communication will then look as follows:

- Client sends a SIP request to the publicly known location of the SIP server (which is actually the location of the SIP redirect server)

- SIP redirect server responds to the client using a 30X redirection response containing the address of the fully functional SIP server
- Client re-sends the SIP request to the address which he obtained from the redirection message
- SIP server processes the request and the SIP session can be initiated normally from now on

However, SIP communication for an attacker using the spoofing mechanism will look as follows:

- Attacker sends a SIP request to the publicly known location of the SIP server (which is actually the location of the SIP redirect server)
- SIP redirect server responds to the spoofed source IP address (so the response never reaches the attacker)

As you can see from the previous examples, the attack can be easily mitigated if the SIP redirect server endures the attack. However, thanks to the simplicity of the SIP redirect server, its implementation is quite simple and straightforward and it is not a problem to implement a redirect server capable of processing tens of thousands of requests per second. Because the fully functional SIP servers are flooded using a few hundreds or thousand requests per second, the attacker is expected to use a similar rate and therefore the SIP redirect server should not have problems to endure such floods. I have implemented a very simple prototype SIP redirect server using Python and Twisted library for my testing purposes. Even though my solution is not optimized at all, it can endure common SIP floods.

6.3.3 Implementation

Even though there are some publicly available implementations of SIP redirect server and all the fully functional SIP servers also offer this functionality, I wanted to demonstrate that implementation of such server is very simple and one does not need complex solutions offering functionalities which are not necessary (these solutions are often superfluously complex). Also, I wanted to have full control over the source code of the application in case that I need to alter it in any way.

Therefore I have used Python (version 2.6) and Twisted library and implemented my own SIP redirect server. The implementation can be found in *prototypes/defense/redirect_server/redirecter.py*. To configure the application, you

can simply edit the global variables defined at the beginning of the source code. To run the application, standard python notation is used (e.g. in terminal type *python redirecter.py*).

The source code is very simple and the source code along with inserted comments should be self-explanatory.

6.3.4 Advantages and Disadvantages

Advantages

Easy implementation The implementation is quite easy and if you do not need extra optimization then a few dozen lines of code in any of higher programming languages can do the trick.

Blocks dumb DoS floods too The generators of simple DoS floods are not able to handle the responses from the targeted server (in this case they cannot handle the SIP redirection response) so they will never reach the actual fully functional SIP server.

Blocks DDoS floods with spoofed sources When an attacker uses source IP address spoofing mechanism, then the attack is completely blocked because the attacker never gets a response with the redirection information.

Disadvantages

Useless against smart attackers If the attacker does the survey of the VoIP environment before deploying the attack and finds the real location of the fully functional SIP server or if the bots creating the attack (without using the source spoofing mechanism) have advanced logic and are capable to react to the redirection responses, the proposed defense is useless.

Need to keep the address of the real SIP server a "secret" If the address of the real SIP server is publicly known, then the attacker will target the server directly.

Additional work for the client Each client has to contact the redirect server at the beginning and then react to the redirection response. This introduces a small delay. However, the delay is so short that it is not observable by a human user.

6.3.5 Improvement Ideas

Even though the solution is usable and effective against attacks with spoofed sources, its use in practical deployment is very limited. This is due to the fact that it is important to keep the location of the actual fully functional SIP server hidden from the potential attacker. Considering the fact that the location is published in all of the generated redirect responses and can be also obtained by capturing SIP traffic of any of the legitimate clients, this becomes almost impossible. Therefore I have analyzed the situation more closely and prepared a few improvement ideas.

The first idea is based upon the decision, which of the redirection responses to use for the SIP redirect server. The most straightforward choice would be "301 Moved Permanently" because a common reaction to this message on the clients side is caching the answer and contacting the new location with every new request. Using this option, every SIP client will have the information about the actual location of the fully functional SIP server stored in its memory, which is not desired. Better choice might be the "302 Moved Temporarily" message. This response is not cached by design and so the location of the SIP server will not be stored anywhere. However, it also means that the client will send the first request for every new session to the redirect server. Probably the best choice would be using a wisely crafted "300 Multiple Choices" message. Using this response, you can easily use a backup SIP server (or servers) and complicate the work of an attacker since he will not get a single target, which he expects.

The second idea introduces usage of a poll of locations for the fully functional SIP server. Instead of using only one instance of a SIP server, you can use more instances with different locations (ideally different machines, but you can use virtualization too) and configure the SIP redirect server such that it will change the location information in the redirection messages it generates according to some pseudo-random algorithm. This should not break the sessions in progress but will redirect new requests to different instances of the SIP server thus mitigating the potential attack.

Another idea is to use the mechanism described for the second one but instead of using a pseudo-random algorithm, you can use load balancing. This would require an implementation of a feedback mechanism such that the instances of the fully functional SIP server will provide the information about their actual load to the SIP redirect server and it would decide which location to insert into the redirection messages based upon the evaluation of the actual loads.

Totally different improvement can be gained if you choose to implement some checks directly on the SIP redirect server. These checks must be lightweight and not consume many resources since it might lead to the situation where the SIP redirect server will become prone to the flood attacks itself. However, checking

whether the SIP messages are well-formed using known patterns for well-formed messages might be a good idea.

It is also possible to use the described defense mechanism just as a part of a more complex defense system. This way, the described defense mechanism can take care of dumb non-flexible attacks with spoofed sources and other parts might take care of other threats. I have used this approach when I combined this defense solution with the following one. The combined solution is described later in this thesis.

6.4 Modular Defense Solution

6.4.1 General Idea

Even though the previously described defense solution using redirection is an interesting solution capable of mitigating many SIP DoS and DDoS attacks, the problem of smart attackers still remains. To check how easy it is to simulate such smart attack, I have tried to create a "smart bot" which will be able to react to the redirection responses and forward his attack flow against the real SIP server. As a basis, I have used the SIPp-DD tool. Even though it is not usable as a single DDoS flood creator in this case (as I have showed, address spoofing is useless against the redirection application) I managed to create a smart DoS flood. Therefore if I have a botnet, I might easily produce a DDoS flood if I run instances of properly configured SIPp-DD on the individual bots. Additionally, I can use a probing bot at first, that might extract the information of the actual location of the SIP server and use it in a consequent DDoS flood with spoofed sources bypassing the redirection defense solution.

Because I showed that smart attackers are a problem, I decided to design a more sophisticated solution. The application described in the following paragraphs is rather a skeleton for creation of a more complex solution than a definitive solution itself. Thanks to its modular design, it is easily extensible and configurable and you might implement and use any heuristics you like.

6.4.2 Architecture

This solution is based upon the idea of preprocessing the SIP messages before they are forwarded to the actual SIP server. The functionality of the preprocessing application is to filter out the messages that can be potentially dangerous to the SIP server (simply said – filter out the attacks). This solution needs two servers - one runs the filtering application and the second runs the actual SIP server. You can use one machine and virtualization, but since the application

itself is expected to consume a fair amount of resources, I cannot recommend that. It is also a good idea to hide the real SIP server inside a private network so it will not be reachable directly from the Internet. This way, you can prevent the possibility that the attacker will bypass your filtering application and attack the SIP server directly. Additionally, you can also guarantee that the computers inside the private network will not be accidentally filtered out by the filtering application (this setup expects that the computers inside the private network are trustworthy). As in the previous approach, the address and port where the application runs is publicly known and referred to as the location of the SIP server.

Now let us have a closer look at the SIP message processing:

- The client creates a SIP request and sends it to the filtering application (which he considers to be the real SIP server).
- The application receives the message.
- The application checks its actual state. There are two possible states - "normal" and "under attack". If the state is set to "normal" then the application forwards the request to the SIP server. If the state is set to "under attack", heuristics are applied to the received request before it is either forwarded to the SIP server or dropped.
- The application computes new state according to the counter of received messages in the last time period (the new state is again either "normal" or "under attack") and the process continues with the next request.

To lighten the load on the application, the responses from the SIP server are routed directly to the client, the application does not handle these in any way. This behaviour was reached using the same mechanism that the attackers might use for DDoS flood attack creation – source spoofing. The filtering application forwards the requests to the SIP server in a transparent way – it spoofs the sources such that the forwarded messages have the source equal to the actual source of the requests and not to the filtering application itself.

I have vaguely mentioned that the application chooses a new state and uses heuristics but I have not described what to imagine under these processes. This is due to the fact that there are several possibilities/configurations for both processes which differ in complexity and usability. I will describe some of the possibilities in the next two sections and also use some of these for the prototype implementation. However, this is by no means a final list of possibilities and you are encouraged to develop your own new methods.

6.4.3 Attack Detection Algorithm

Flood attacks are generally easily detectable since they are composed of huge amounts of messages whose purpose is to congest the target service. SIP DoS and DDoS floods do not differ in this characteristic and thanks to the fact, that SIP is used mainly for initiation and termination of sessions only, it is very unusual that high amounts of messages are processed in a time period. Therefore it should not be a problem to define an absolute threshold of SIP messages processed by a SIP server per time period in any SIP environment which corresponds to the common traffic. Choice of concrete values for the threshold and time period may vary according to the individual needs of the concrete SIP environment.

After you setup the threshold and time period, the process of state decision is quite straightforward. One counter is used and increased per every received request. The counter is periodically zeroed with every passed time period. Two checks are periodically done. First check is whether the counter exceeds the threshold. It is done after each received request and if the counter exceeds the threshold then the application changes state from "normal" to "under attack". The second check is done at the beginning of every time period before the counter is zeroed. If the actual value of the counter is higher than the threshold, the application remains in the "under attack" state. If the actual value of the counter is lesser than the threshold and the application was in the "under attack" state, it changes its state to "normal". Otherwise it remains in the "normal" state.

There are a few possible improvements of this decision mechanism. You can use a varying time period which will reflect the actual situation (shorter under attack, longer otherwise...absolute or proportional relative changes etc.). You can use a varying threshold which will reflect the actual load. You can introduce some analysis over a longer time (including more time periods) and use its results for changing the mentioned parameters. However, you must keep in mind, that very long time period can lead to late detection of an attack, which might affect the processing capabilities of the SIP server and very short time periods might exhaust the resources of the server running the application. A good recommendation is to do some analysis of the logs of SIP traffic in your network and setup the threshold and time period according to these and after that start experimenting with varying values.

A notable improvement might be using more counters for different types of SIP requests. This will introduce some additional load on the server running the application, but the overall results might be better since the floods are generally uniform (using only one type of request) and only a few types of SIP requests are suitable for a flood. This might also mitigate the problems if you use some SIP messages for additional services (like SIP INFO for additional data transfer) and these messages will make your threshold too high.

6.4.4 Filtering Approaches

The heuristics will be used in the application only when there is a suspicion that the attack is in progress (the application is in the "under attack" state). The purpose of the heuristics is to decide whether the message should be forwarded to the SIP server or dropped. There are numerous ways how to acquire this decision. The ones I considered are:

Access lists The access lists may be based either upon IP address of the client or upon provided credentials. You might use classic whitelist, graylist or blacklist. A better option might be a combination of whitelist and list containing logs of usage in the last few days. There are numerous possibilities and none is generally the best...it depends on the concrete SIP environment.

Message type control Since the SIP DDoS floods will be often uniform in the type of the message, it might be useful to block only that type of the message and let the rest come through. This might be a problem if the message type used is INVITE but might be very useful if the message type is REGISTER – already registered clients will be fine and may continue using the service without problems.

Message payload control Supposedly, the messages used in a SIP DDoS floodl have a common pattern. Either it might be only a duplication of the exactly same message or there might be a part generated randomly – these patterns can be tracked and used as blocking patterns for attack messages. Using this approach might be a bit complicated since you must keep in mind not to make the process too costly. Otherwise the application can become the congested one and the service will be disrupted even though the SIP server will not be harmed.

A good idea for future research is usage of neural networks to monitor the SIP traffic and "automatically" learn what is legitimate and what is not. Using tools like CAMNEP [23] to learn these patterns might be also very useful and I will definitely try it once I acquire an advanced testing site with legitimate SIP traffic.

6.4.5 Implementation

My goal was simple implementation providing all the described basic functionality, therefore I have used Python (version 2.6) with libdnet and dpkt libraries. The libdnet library is used to access the network services, dpkt library is used for IP and UDP header creation.

The application is multi-threaded (well, two-threaded to be precise). One thread represents a timer and executes the periodic state checks and state changes, if necessary. The other thread is the main thread of the application that handles message receive, uses heuristics on the received messages if necessary and forwards the messages towards the SIP server. The forwarding mechanism is implemented transparently using the RAW socket for forwarded messages and explicit crafting of IP and UDP headers such that the forwarded message still looks like it was sent by its original sender and not re-send by the application.

The application can be found in *prototypes/defense/filter_app/alfinal.py*. To configure the application, you can simply edit the global variables defined at the beginning of the source code. To run the application, standard python notation is used (e.g. in terminal type *python alfinal.py*).

6.5 Combined Defense Solution

The two already presented defense solutions are based upon different approaches and aim to mitigate slightly different types of attacks. I thought that it might be interesting to combine both of these solutions and create a double-layered defense solution. The diagram of the SIP traffic for the combined solution is described in Fig. 6.1.

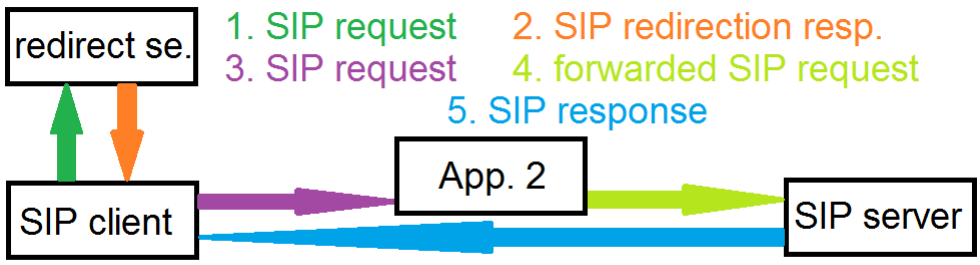

Figure 6.1: SIP traffic diagram

As you can see, the combined solution is only serialization of both principles. At first, the client contacts the redirect server and gets the redirection message. This step will filter out all messages with spoofed sources, which is very useful because this way, the second application does not have to process them. The client then follows the redirection and contacts the second application, which applies its processing and the message is finally forwarded to the SIP server.

This solution might look a bit ping-pong like, since the message is being send, resend and forwarded, but the tests show that the delay introduced by

this additional processing is relatively small and the results in terms of blocking various SIP DoS and DDoS flood are satisfiable.

Chapter 7

Tests

This chapter describes some of the tests I have done during my research. The tests aim at proving the ideas which I have stated and at proving the usability of the SIPp-DD tool and proposed defense solutions. To maintain understandableness, every test will be described the same way including its goal, initial SIP environment setup, details about the test process and summary of the results.

7.1 General Settings

Every test simulates a SIP DoS or DDoS attack. SIPp-DD is used for simulation of the majority of attacks to prove its usability for simulation of various types of attacks. Asterisk PBX (version 1.6.0.21) implementation of SIP server is used as the target in majority of the tests. Asterisk was chosen because it is a very popular implementation used in many small and medium-sized companies. If not stated otherwise, Asterisk uses the database of 1000 legitimate users stored in its internal database (using the *sip.conf* file). I have also done some tests using the OpenSIPS implementation which is a very flexible and suitable solution even for bigger companies with thousands of SIP clients. However, the general assumptions do not differ for the two mentioned implementations (and neither do they for any other implementation). The only difference is in required number of packets involved in the attack (for flooding attacks) and in the actual point of failure (for Asterisk this is often CPU exhaustion, for OpenSIPS is typical memory exhaustion or insufficient capacity of underlying services providing the user database).

In many tests, a simulation of a legitimate SIP client trying to contact the targeted SIP server with a valid registration request is used. This is used to check the impact of the attack on the legitimate clients. SIPp call generator is used for this purpose. It is configured to send one register request per second

and, if the server replies, successfully finish the registration handshake. I am aware that this method is very rough and cannot be used for precise detection of the impact, however it provides the necessary basic feedback – whether the SIP service is affected by the attack or not. The described probing mechanism using a legitimate client simulation is referred as "legitimate client" in the tests. It might be very interesting to do the simulations in a more complex SIP testbed ideally with some common SIP traffic to measure the actual real-like impact of the attacks. However, I do not have resources to do such research. This will hopefully be addressed in some future work.

To monitor system resources on the target machine during the attack, a unix monitoring utility called *top* is used. This tool monitors both the CPU and memory usage per each running process. I am interested in the resource consumption produced by the SIP server implementation (Asterisk) during my tests. Because *top* does not scan the system continuously but probes the system resources periodically, I have decided to set the granularity to 1.0 second (e.g. *top* does a snapshot of the resource consumption once per second). The captured results are then filtered using *awk* and analyzed using Matlab.

7.2 Testbed

The testbed is composed of three machines interconnected in a private network. To interconnect the machines, an Asus 4-port switch with maximum speed of 100Mbit/s is used. The main machine is called *Odin* and is used for running the SIP server instances, which are targeted during the simulations. The other two machines are called *Thor* and *Loki* and are used for simulation of the attacker and the legitimate client respectively. The hardware configurations are as follows:

Odin

CPU Intel Xeon Quadcore @ 2.50Ghz

Memory 2x4GB DIMM @ 667Mhz

OS CentOS 5.4

Thor, Loki

CPU Intel Xeon Dual-core @ 3.00Ghz

Memory 4GB DIMM @ 667Mhz

OS CentOS 5.4

7.3 General Test Pattern

Goal of the test Description of what I want to demonstrate (prove) in the test.

Testbed Description of the SIP testbed – involved SIP components.

Attack setup Used attack simulation tool and its configuration.

Defense setup SIP defense solution deployed on the targeted SIP component.

Test timeline Length of the test, start and stop times of the attack.

Test process Description of the actual process of the test – changing states of the target, attacker, defense, resource consumption etc.

Results Summary of the test.

7.4 Test Suites

7.4.1 Initial Benchmarks

DoS Flood Attack

Goal of the test Prove that Asterisk is prone to the DoS flood attack when there is no defense solution deployed to protect it.

Testbed Asterisk, legitimate client, SIPp-DD.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target using attack rate 500mps (messages per second). Attack duration is 20s.

Defense setup None.

Test timeline Test duration is 35s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process The test progressed as expected. At first, there was minimal load on the CPU and the requests from the legitimate user were processed normally. Then, after the attack began, the load on the CPU increased gradually in a few seconds and once it got to 100% load, the requests from the legitimate user started to be retransmitted since there was no response from the server (I could have easily changed the behaviour of the legitimate user not to retransmit the requests, however, I did not intend to since retransmission is a common behaviour of SIP clients). I have observed one request from the legitimate client that was processed during the attack, however since the registration is a two-step process, the second part of it did not have the luck and began to be retransmitted too. Right after the attack ended and the CPU load decreased, the requests being retransmitted were answered at once (if the attack took longer, the retransmissions will eventually time-out and the requests will fail). The process of the attack can be clearly seen in Fig. 7.1 representing the CPU load during the attack.

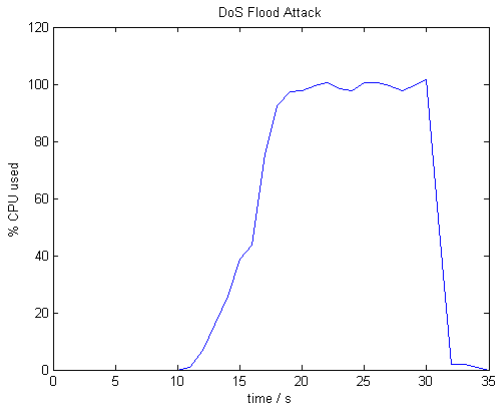

Figure 7.1: CPU load on the target SIP server during a DoS attack.

Results The test showed that Asterisk is very prone to DoS flood attacks, 500mps was enough to completely block the service.

DDoS Flood Attack

Goal of the test Prove that Asterisk is prone to the DDoS flood attack when there is no defense solution deployed to protect it.

Testbed Asterisk, legitimate client, SIPp-DD.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target using attack rate 500mps (messages per second) with active source spoofing mechanism. A set of 1000 different IP addresses is used as the spoofed sources. Attack duration is 20s.

Defense setup None.

Test timeline Test duration is 35s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process The test progress was very similar to the one described in the previous test simulating a DoS flood attack. This was expected since the attacks differ only in the source IP addresses in the packets delivering the SIP messages and that does not affect the message processing. To demonstrate the process of the attack I provide the graph representing the CPU load which can be seen in Fig. 7.2.

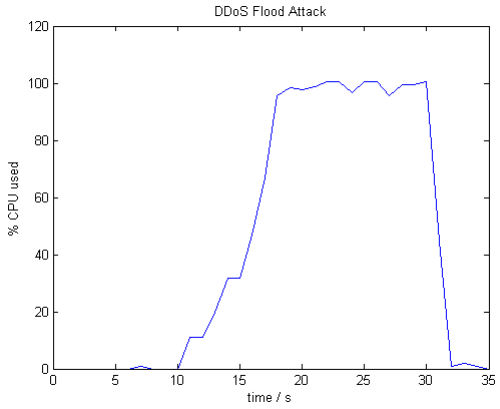

Figure 7.2: CPU load on the target SIP server during a DDoS attack.

Results The test showed that Asterisk is very prone to DDoS flood attacks, 500mps was enough to completely block the service.

Malformed Message Attack – Basic

Goal of the test Prove that Asterisk is able to handle all the well known types of malformed messages used in the c07-sip part of the PROTOS project [15].

Testbed Asterisk, c07-sip-r2.jar

Attack setup The full set of 4527 known test cases prepared in the c07-sip testing tool were sequentially send to the Asterisk.

Defense setup None.

Test timeline The whole test took about 70 minutes.

Test process After elimination of a problem caused by wrong value of the LANG environment variable (solution available on the c07-sip webpage) the test ran smoothly, without errors. The resource consumption was very low throughout the whole test.

Results Asterisk did not crash, all the test cases ended successfully. This proves that Asterisk is protected against the well known types of malformed messages and is able to handle them.

Malformed Message Attack – SIPp-DD

Goal of the test Prove that SIPp-DD is able to generate attacks of malformed message type.

Testbed Asterisk, SIPp-DD.

Attack setup I have generated a file full of pseudo-random rubbish (mixture of characters generated using */dev/urandom*, *base64* and *grep*). Then I have inserted the keyword *SEQUENTIAL* at the beginning of the file and used this file as an external CSV providing values for the XML file defining the scenario for the SIPp-DD. The scenario is very simple – just send a common REGISTER request and replace the fields expecting the username values using the values taken from the CSV. The attack rate was set to 10mps. Malformed requests then looks like the following example:

```
REGISTER sip:4u30+Grd8LKnP4zcphk4VHTBTwjW3bT8PtjdVFhjDHBbJJtbAMO/Z
8VFfBf4rliwlGQjoeu6M36k SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK-9901-1-0
From: <sip:4u30+Grd8LKnP4zcphk4VHTBTwjW3bT8PtjdVFhjDHBbJJtbAMO/Z8V
FfBf4rliwlGQjoeu6M36k@>;tag=1
To: <sip:4u30+Grd8LKnP4zcphk4VHTBTwjW3bT8PtjdVFhjDHBbJJtbAMO/Z
```

```
8VFfBf4rliwlGQjoeu6M36k@4u30+Grd8LKnP4zcphk4VHTBTwjW3bT8PtjdV
FhjDHBbJJtbAMO/Z8VFfBf4rliwlGQjoeu6M36k>
Call-ID: 1-9901@127.0.0.1
CSeq: 1 REGISTER
Contact: sip:4u30+Grd8LKnP4zcphk4VHTBTwjW3bT8PtjdVFhjDHBbJJtbAMO/Z
8VFfBf4rliwlGQjoeu6M36k_z9hG4bK-9901-1-0@127.0.0.1:5060
Max-Forwards: 5
Expires: 3600
User-Agent: SIPp/Linux
Content-Length: 0
```

Defense setup None.

Test timeline Test duration is 10 minutes.

Test process As in the previous test using the malformed messages from c07-sip, the progress of the test was smooth without errors or distinct resource consumption.

Results Asterisk was able to handle all the 6000 requests without crash or failure. Note that this does not prove that Asterisk is protected against the malformed message attacks, I only wanted to demonstrate that it is quite easy to simulate such attack using SIPp-DD.

7.4.2 DoS and DDoS Attacks Using Different SIP Messages

Different SIP message types are processed differently by the SIP server. The purpose of these simulations is to demonstrate that some SIP message types are more suitable to be used in a flood attack e.g. messages that consume most of servers resources when being processed (in case of Asterisk implementation, I focus on the CPU load). Because all the simulations in this section share the same settings, I do not use the general test pattern for the description of each simulation. The shared settings are:

Goal of the test Show the resource consumption (CPU load) produced by processing of the attack composed of the specified SIP message types.

Testbed Asterisk, SIPp-DD.

Attack setup SIPp-DD is configured to send 50000 SIP messages (of the specified type) with invalid credentials to the target using attack rate 500mps. Attack duration is 100s.

Defense setup None.

Test timeline Test duration is 150s, attack begins simultaneously with the test and ends 50s before the end of the test.

To compare the results, the graph describing the CPU load during the individual tests is used. Each test is named according to the message type (or types) it uses. If there are more message types used in one test then their distribution in the test is uniform (if 500 messages are sent in a test named INVITE + REGISTER then these 500 messages are composed of 250 INVITE and 250 REGISTER messages which are periodically changing in the sequence).

INVITE

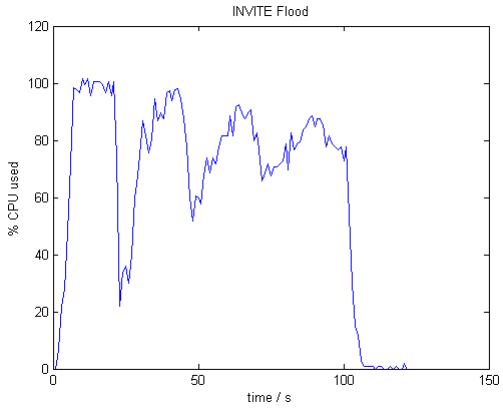

Figure 7.3: CPU load on the target SIP server during a DoS attack using INVITE messages.

You can see that the CPU was not fully overloaded, but even though the legitimate traffic was badly affected. The phenomenon of jumping CPU usage when processing SIP INVITE messages is present in every test using SIP INVITE messages and might be worth closer inspection (interesting fact is that this phenomenon is not present in any other message processing phase, only by INVITE message).

REGISTER

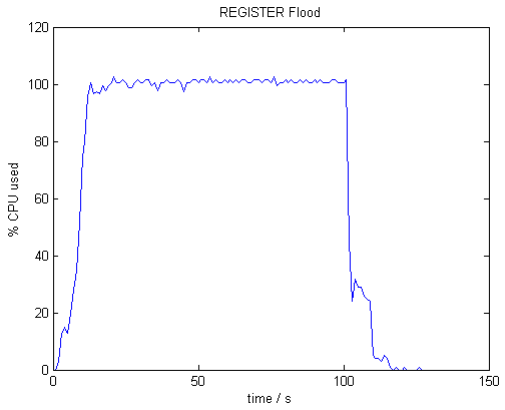

Figure 7.4: CPU load on the target SIP server during a DoS attack using REGISTER messages.

You can see that the CPU was overloaded, legitimate traffic was badly affected.

BYE

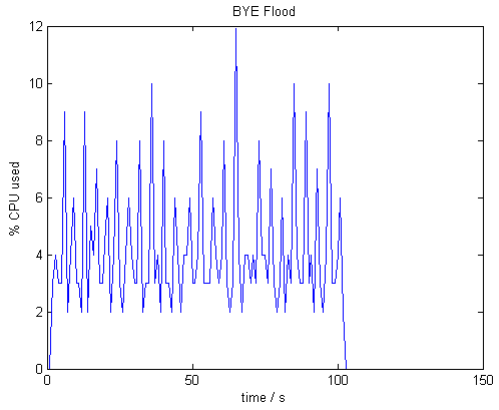

Figure 7.5: CPU load on the target SIP server during a DoS attack using BYE messages.

You can see that the CPU had plenty of capacity left. The legitimate traffic was not affected at all.

ACK

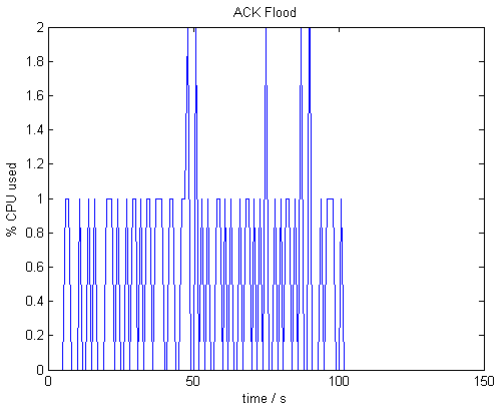

Figure 7.6: CPU load on the target SIP server during a DoS attack using ACK messages.

You can see that the CPU had plenty of capacity left. The legitimate traffic was not affected at all.

CANCEL

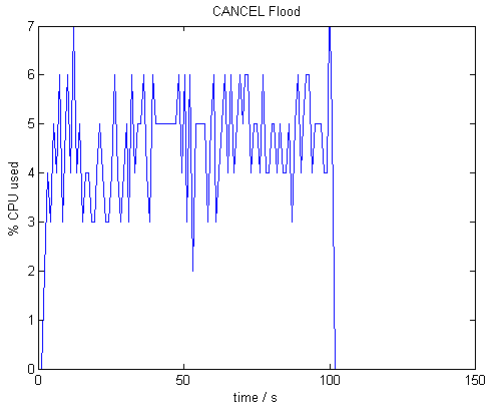

Figure 7.7: CPU load on the target SIP server during a DoS attack using CANCEL messages.

You can see that the CPU had plenty of capacity left. The legitimate traffic was not affected at all.

OPTIONS

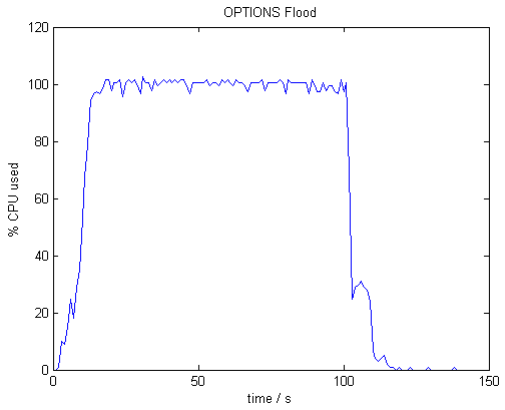

Figure 7.8: CPU load on the target SIP server during a DoS attack using OPTIONS messages.

You can see that the CPU was overloaded, legitimate traffic was badly affected.

SIP 200 OK

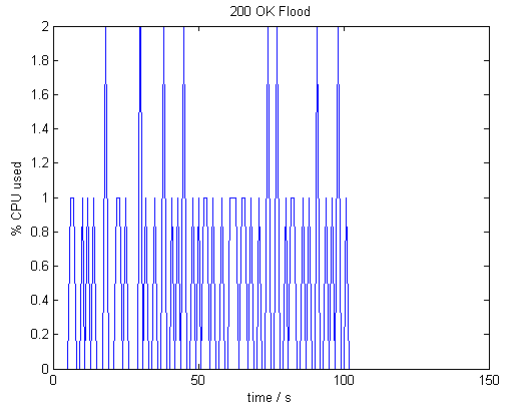

Figure 7.9: CPU load on the target SIP server during a DoS attack using 200 OK messages.

You can see that the CPU had plenty of capacity left. The legitimate traffic was not affected at all.

REGISTER + INVITE

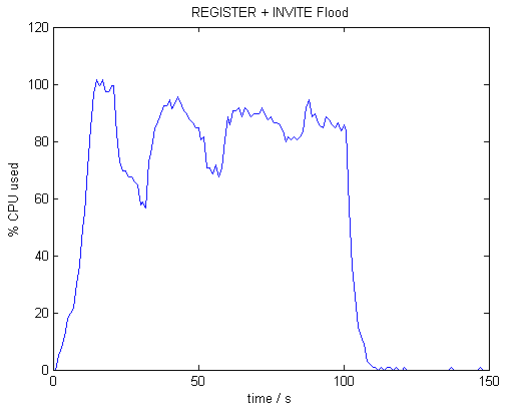

Figure 7.10: CPU load on the target SIP server during a DoS attack using the combination of REGISTER and INVITE messages.

You can see that the CPU was not fully overloaded, but even though the legitimate traffic was badly affected.

REGISTER + INVITE + OPTIONS

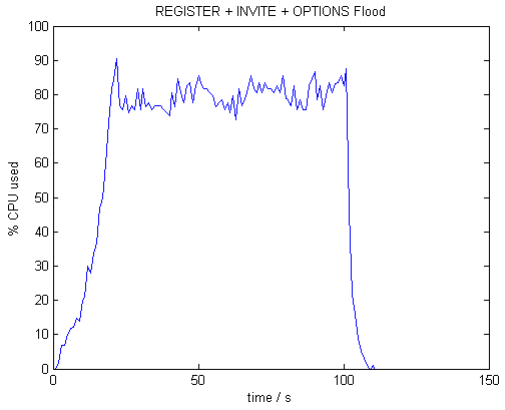

Figure 7.11: CPU load on the target SIP server during a DoS attack using the combination of REGISTER, INVITE and OPTIONS messages.

You can see that the CPU was not fully overloaded, but even though the legitimate traffic was badly affected.

REGISTER + INVITE + ACK + BYE + 200 OK

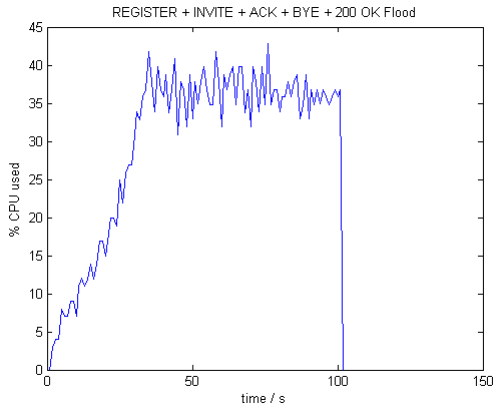

Figure 7.12: CPU load on the target SIP server during a DoS attack using the combination of REGISTER, INVITE, ACK, BYE and 200 OK messages.

You can see that the CPU had about half of its capacity left. The legitimate traffic was not affected at all.

Results

As you can see in the previous tests, message types INVITE, REGISTER and OPTIONS are convenient to be used in flood attacks. The tests also showed that combinations of message types in an attack does not improve the impact of the attack on the target SIP server. Even though I have tested only two SIP server implementations (Asterisk, which I used in the described tests and OpenSIPS, which I used in other parts of my research) I can suppose that the resulting impact will be higher for the attacks using the three already mentioned message types on other implementations too (due to the processing mechanism of these messages that must be similar). Proving this assumption might be interesting for the future research.

7.4.3 Basic DoS Defense

DoS – Iptables Rule

Goal of the test Prove that the approach using simple blocking rule for Iptables with the *recent* module works against SIP DoS flood attacks.

Testbed Asterisk, legitimate client, SIPp-DD.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target using attack rate 500mps (messages per second). Attack duration is 20s.

Defense setup Iptables is configured to block the source IP if the number of packets arriving from that IP to the SIP port exceeds the limit of 100 packets per 10 seconds. The duration of the block is set to 10 minutes.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process The requests of the legitimate client were processed without problems, not a single delay observed. If you have a look at the CPU load graph 7.13, you can see only a very small peak at the beginning of the attack (corresponds to the first one hundred packets that actually reached Asterisk). This is thanks to the fact that Iptables itself is very fast and reacts immediately (not a single packet over the threshold is let in). The second peak corresponding to one percent load at 18th second of the test is not significant – it sometimes happen that Asterisk takes a one percent CPU load even if idle.

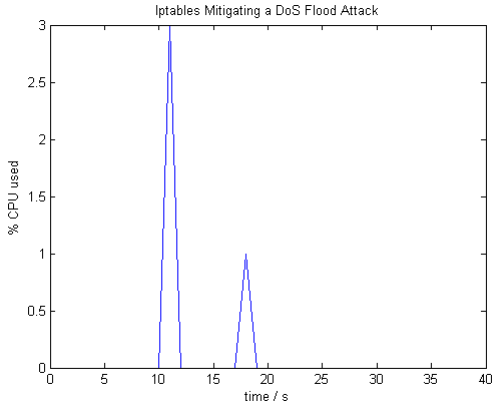

Figure 7.13: CPU load on the target SIP server during a DoS attack with active Iptables defense solution.

Results The DoS attack was successfully mitigated, the requests of the legitimate client were processed without any delay.

DoS – Snort + Snortsam + Iptables

Goal of the test Prove that the approach using the combination of Snort, Snortsam and Iptables works against SIP DoS flood attacks.

Testbed Asterisk, legitimate client, SIPp-DD.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target using attack rate 500mps (messages per second). Attack duration is 20s.

Defense setup Snort is configured to watch for packets containing SIP messages of defined type (specifically for types suitable for flood attacks – INVITE, REGISTER and OPTIONS). If the number of packets from one source IP address exceeds the limit of 20 in one second, alert is send to the Snortsam plugin and it initiates a blocking rule for Iptables. The duration of the block is set to 10 minutes.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process Since the solution is a combination of tools, I expected it to react slower than the previous solution based upon Iptables itself. This expectation was fulfilled but additionally, I have observed an interesting fact – it depends whether Snort detects the attacker for the first time (after fresh restart) or repeatedly. The situation concerning CPU load in case of fresh start can be seen in Fig. 7.14. Supposedly there must be some mechanism inside Snort that caches the addresses that have violated the rules before because in the situation of the repeated attack, the reaction of Snort is almost the same as in case of the previous solution (using only Iptables itself). Even though it did not show in this test, note that Snort itself can consume a lot of resources when the traffic flow it scans is high and you should consider this when planning to use it.

Results The DoS attack was successfully mitigated, the requests of the legitimate client were processed without any delay.

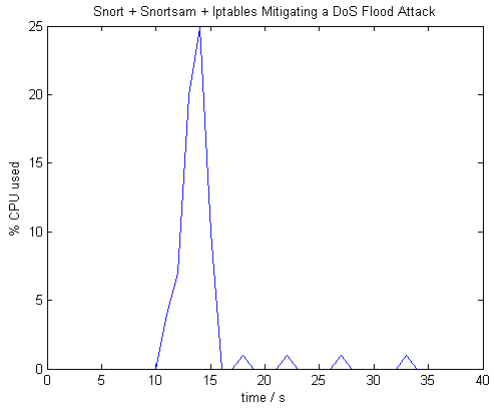

Figure 7.14: CPU load on the target SIP server during a DoS attack with active Snort + Snortsam + Iptables defense solution.

DoS – Fail2Ban + Iptables

Goal of the test Prove that the approach using the combination of Fail2Ban and Iptables works against SIP DoS flood attacks.

Testbed Asterisk, legitimate client, SIPp-DD.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target using attack rate 500mps (messages per second). Attack duration is 20s.

Defense setup Fail2Ban is configured to watch the log of the SIP server. Once it detects five unsuccessful SIP requests (dropped because of wrong format, containing invalid credentials etc.) from one source IP address it initiates a blocking rule for iptables. The duration of the block is set to 10 minutes.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process Because Fail2Ban is in general only a log parser and the log is being checked periodically each second, there is a small delay introduced. This delay can be detected in the graph of the CPU load – you can see a small peak at the beginning of the attack in Fig. 7.15. However, this delay did not affected the

continuity of the regular SIP flow and all the requests were processed without delay.

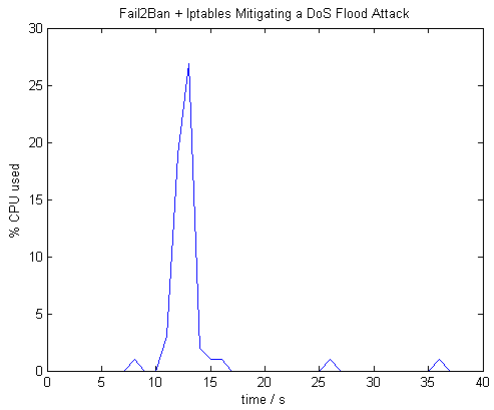

Figure 7.15: CPU load on the target SIP server during a DoS attack with active Fail2Ban + Iptables defense solution.

Results The DoS attack was successfully mitigated, the requests of the legitimate client were processed without any delay.

DDoS

Goal of the test Demonstrate, that all the proposed DoS defense solutions are useless against a DDoS attack.

Testbed Asterisk, legitimate client, SIPp-DD.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target using attack rate 500mps (messages per second) with active source spoofing mechanism. A set of 1000 different IP addresses is used as the spoofed sources. Attack duration is 20s.

Defense setup All the three mentioned basic DoS defense solutions were used in different tests. The concrete numbers used in this test process description were taken from the capture where the solution based upon combination of Snort, Snortsam and Iptables was used. The process and result for the remaining two solutions are almost the same.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process The process is very similar to the process of the DDoS flood without any defense solution deployed (actually, the situation is the same since the defense solution is unable to mitigate the attack). The corresponding graph of the CPU load can be seen in Fig. 7.16. Once the CPU load reached 100%, the requests from the legitimate client began to be retransmitted and they were successfully processed only after the attack ended. Interesting is the small fluctuation visible around the center of the attack. This fluctuation is due to the fact that Snort itself takes some resources (even though the flow composed of 500 packets per second is relatively small, Snort takes about 2% of the CPU load) as I have already mentioned. If the flow was higher (thousands of packets per second), Snort would take noticeable amount of CPU load and it might complicate the processing even more. Therefore it is wise to situate Snort on another machine than the SIP server itself.

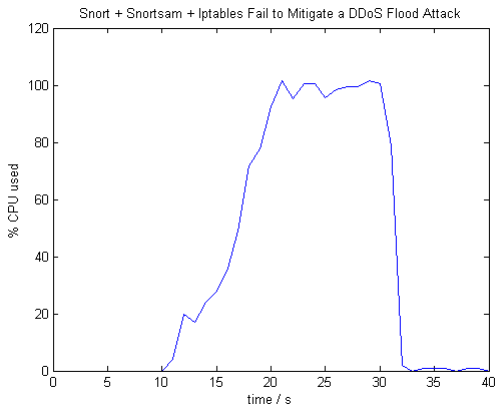

Figure 7.16: CPU load on the target SIP server during a DDoS attack with active Snort + Snortsam + Iptables defense solution.

Results The DDoS attack was not mitigated at all.

7.4.4 Advanced DoS and DDoS Defense

Dumb DoS – Redirection Defense Solution

Goal of the test Prove that the defense solution based upon the redirection mechanism (described in chapter *New Defense Solutions*) is capable of mitigating a dumb DoS attack (dumb means that the attackers tool/script is unable/not configured to handle the redirection messages).

Testbed Asterisk, legitimate client, SIPp-DD, redirection script. Asterisk is running on *odin*, both legitimate client and redirection script are running on *thor*. SIPp-DD is running on *loki*. This distribution was chosen because I did not want to run both Asterisk and redirection script on the same machine since it may affect the results (distort the delay etc.)

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target (in this case the target is the redirection script, not the actual SIP server) using attack rate 500mps (messages per second). The scenario for SIPp-DD contains only one *send* element to send the initial request and does not expect any reply. Attack duration is 20s.

Defense setup The redirection script is the target of the attack instead of the actual SIP server. It is configured to reply to each SIP request with a redirection response containing the location of the actual SIP server.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process All the requests of the legitimate client are processed without delay. Since the messages composing the attack will never reach Asterisk (each attack request arrives to the redirection script and a redirection message is send as a response... however, SIPp-DD scenario is not configured to handle this message), there is no increase in resource consumption nor other affection of the SIP server.

Results The proposed solution is capable of mitigating a dumb DoS attack.

Smart DoS – Redirection Defense Solution

Goal of the test Show that the defense solution based upon the redirection mechanism is not able to mitigate a smart DoS attack (smart means that the attackers tool is able to handle the redirection messages and can redirect the attack flow to the actual SIP server).

Testbed Asterisk, legitimate client, SIPp-DD, redirection script. Asterisk is running on *odin*, both legitimate client and redirection script are running on *thor*. SIPp-DD is running on *loki*.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target (in this case the target is the redirection script, not the actual SIP server) using attack rate 500mps (messages per second). The scenario for SIPp-DD contains a *send* element to send the initial request at the beginning and then an action specifying that if the redirection response arrives, the message should be re-send to the target contained in the redirection message payload. Attack duration is 20s.

Defense setup The redirection script is the target of the attack instead of the actual SIP server. It is configured to reply to each SIP request with a redirection response containing the location of the actual SIP server.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process The situation is very similar to the process described for the DoS flood attack test without any active defense (because it is in fact the same from the point of the SIP server). Because the messages are first send to the redirection server and then re-send to the actual SIP server, the whole attack flow reaches its target and the actual SIP server is flooded. The CPU load graph (at *odin*) is therefore very similar to the one already provided for the DoS flood attack without active defense in Fig. 7.1.

Results The simulation shows that the defense solution based upon the redirection mechanism cannot mitigate smart DoS flood attack.

DDoS – Redirection Defense Solution

Goal of the test Prove that the defense solution based upon the redirection mechanism is able to mitigate DDoS flood attack produced by a tool using the source spoofing mechanism.

Testbed Asterisk, legitimate client, SIPp-DD, redirection script. Asterisk is running on *odin*, both legitimate client and redirection script are running on *thor*. SIPp-DD is running on *loki*.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target (in this case the target is the redirection script, not the actual SIP server) using attack rate 500mps (messages per second). The spoofing mechanism in SIPp-DD is active and is using a set of 10000 different IP addresses so that no address repeats in the attack. The scenario for SIPp-DD contains a *send* element to send the initial request at the beginning and then an action specifying that if the redirection response arrives, the message should be re-send to the target contained in the redirection message payload. Attack duration is 20s.

Defense setup The redirection script is the target of the attack instead of the actual SIP server. It is configured to reply to each SIP request with a redirection response containing the location of the actual SIP server.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process The situation is the same as in the simulation of a dumb DoS attack, only the reason why the packets do not reach the actual SIP server differs. In this simulation, the request arrives at the redirection script, is processed and the redirection reply is send to the source address specified in the packet that brought the request. This address is spoofed, so the response never reaches the attackers tool. Therefore there is no increase in resource consumption nor other affection of the SIP server.

Results The proposed solution is capable of mitigating a DDoS flood attack generated using a source spoofing mechanism.

DoS – Modular Defense Solution

Goal of the test Prove that the modular defense solution (described in chapter *New Defense Solutions*) is capable of mitigating a DoS flood attack (both dumb and smart as they were described in the previous simulations, because it does not make a difference for this solution).

Testbed Asterisk, legitimate client, SIPp-DD, modular defense script. Asterisk is running on *odin*, both legitimate client and modular defense script are running on *thor*. SIPp-DD is running on *loki*.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target (in this case the target is the defense script, not the actual SIP server) using attack rate 500mps (messages per second). Attack duration is 20s.

Defense setup The script is configured as follows: time period 1s, threshold per time period 100 messages, attacker is not whitelisted.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process At the beginning of the attack, you can observe a small peak in the CPU load graph in Fig. 7.17. This peak is produced by the first batch of SIP messages that actually reached the SIP server (the first second when the defense script was still in "normal" state and the packet count exceeded the threshold). You can again observe some peaks corresponding to one percent load of the CPU. These are common for Asterisk even in idle state and I do not consider them relevant. After the defense script switched to the "under attack" state, no more attack packets were forwarded to the SIP server and so the load returned to insignificant values. The requests from the legitimate client were processed without delay during the whole test.

Results The proposed solution is capable of mitigating a DoS flood attack.

DDoS – Modular Defense Solution

Goal of the test Prove that the modular defense solution (described in chapter *New Defense Solutions*) is capable of mitigating a DDoS flood attack.

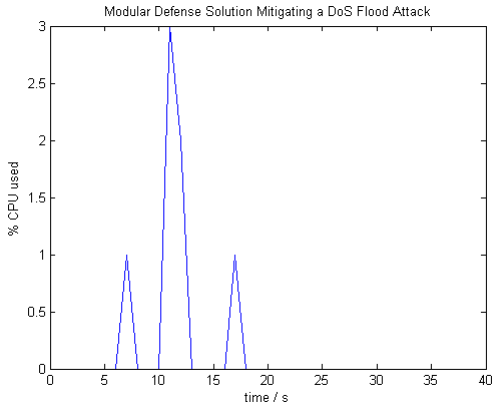

Figure 7.17: CPU load on the target SIP server during a DoS attack with active modular defense solution.

Testbed Asterisk, legitimate client, SIPp-DD, modular defense script. Asterisk is running on *odin*, both legitimate client and defense script are running on *thor*. SIPp-DD is running on *loki*.

Attack setup SIPp-DD is configured to send 10000 SIP REGISTER messages with invalid credentials to the target using attack rate 500mps (messages per second) with active source spoofing mechanism. A set of 10000 different IP addresses is used as the spoofed sources. Attack duration is 20s.

Defense setup The script is configured as follows: time period 1s, threshold per time period 100 messages, attacker is not whitelisted.

Test timeline Test duration is 40s. Legitimate client is activated 5 seconds after the beginning of the test. Attacker is activated 10 seconds after the beginning of the test.

Test process The situation is similar to the previous simulation, the fact that it is the DDoS flood now does not make any difference for the defense solution. The graph of the CPU load is available in Fig.7.18. The difference in the load from the previous simulation is caused by the defense solution design – it depends how many packets (SIP messages) arrive in the first second, when the attack is detected. Only these packets are forwarded to the SIP server, the others are dropped (after the script switches to the "under attack" state). The requests from the legitimate client were processed without delay during the whole test.

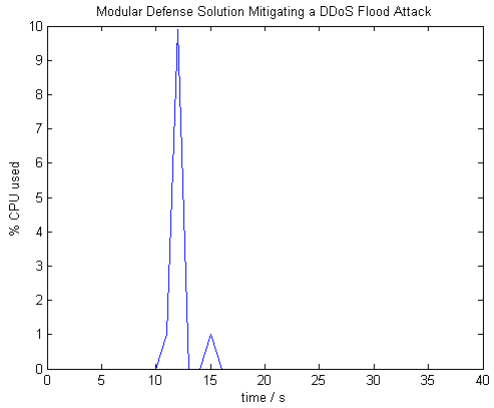

Figure 7.18: CPU load on the target SIP server during a DDoS attack with active modular defense solution.

Results The proposed solution is capable of mitigating a DDoS flood attack

Chapter 8

Conclusion

In this thesis, I focused on the area of SIP security. Because of the fast expansion of the VoIP telephony, which is most often based upon SIP, and the lack of proper defense mechanisms in plenty of deployed SIP environments, I consider the topic to be of utmost importance and find it necessary to point out the major threats and propose ideas usable in practical defense against them.

Firstly, I offered a short introduction into SIP for these, who are not accustomed with it, and defined the terms "DoS attack" and "DDoS attack". Afterwards, I provide the analysis of the DoS and DDoS attacks targeting SIP from the global perspective. I proposed a classification for the attacks targeting SIP. The classification tries to divide different types of attacks according to their most important aspects. One of the criteria is distributiveness, which showed to be a fundamental aspect for the design of defense solutions. I provide some ideas and designs that can be used when creating a defense solution against the major attack types. I have also summarized the information about other projects focusing on SIP security, which I encountered during my research and discussed their basic ideas and approaches.

In the next chapter, I focused on the practical realization of attacks against SIP environments. I have demonstrated how easy it is to deploy such attack using publicly available tools and the typical results of such attacks for the targets. To make the testing of SIP environments easier, I propose my testing tool, which I called SIPp-DD. It is a modified version of a very popular call generator SIPp and can be easily used to simulate any of the major SIP-specific DoS and DDoS attack types discussed in this thesis. Important aspect of SIPp-DD is the IP address spoofing mechanism, that is used for simulation of DDoS attacks.

Once I was able to successfully simulate attacks, I have switched to the defense topic. The already mentioned IP spoofing mechanism proved to be vital for the defense solution design and I granted it a notable portion of my research

time. The two new defense solutions proposed in this thesis both consider the usage of the IP spoofing mechanism on the attackers side. The first defense solution is tailored specifically to mitigate attacks using this mechanism, the second defense solution is rather a skeleton (in terms of design and idea), which can be used to implement a fully-pledged defense solution.

The final chapter consists of description of practical tests and simulations demonstrating the important aspects of different attack types and defense solutions. My aim was to prove the theoretical assumptions which I have made throughout the thesis as well as to stress the problems and weaknesses that might appear in real SIP deployments.

As for the future work, there are numerous possibilities. It would surely be useful to improve the SIPp-DD tool – improve the time management, add a framework for synchronization of more parallel instances, prepare more simulation scenarios etc. More simulations deployed in a real SIP environment or in an advanced simulated SIP environment consisting of more different SIP components and containing also a common SIP traffic might lead to new ideas for defense solution design (especially in the area of synchronization and distributiveness). Implementation of a fully-pledged defense solution based upon the skeleton which I proposed in the thesis is also an interesting option.

Bibliography

[1] Gayraud R., Jacques O. et al.: SIPp traffic generator for the SIP protocol, *http://sipp.sourceforge.net*

[2] Camp K. (2003): *IP Telephony Demystified*. The McGraw-Hill Professional.

[3] H.323 Standards, *http://www.packetizer.com/ipmc/h323/standards.html*

[4] Request for Comments 2543 - SIP: Session Initiation Protocol, *http://www.ietf.org/rfc/rfc2543.txt*

[5] Request for Comments 3261 - SIP: Session Initiation Protocol, *http://www.ietf.org/rfc/rfc3261.txt*

[6] SIP: Session Initiation Protocol - complete reference, *http://www.networksorcery.com/enp/protocol/sip.htm*

[7] Zar J. et al.: VOIPSA - VoIP Security and Privacy Threat Taxonomy, *http://www.voipsa.org/Activities/taxonomy.php*

[8] Endler D., Collier M. (2007): *Hacking Exposed VoIP: Voice Over IP Security Secrets & Solutions*. McGraw-Hill.

[9] Low Cost Tools for Secure and Highly Available VoIP Communication Services, *http://www.snocer.org*

[10] SecSip - Open Source SIP Networks Protection System, *http://secsip.gforge.inria.fr*

[11] Nassar M., Nicollini S., State R., Ewald T. (2007): *Holistic VoIP Intrusion Detection and Prevention System*. IPTCOMM.

[12] Groven A. K. et al. (2010): EUX2010SEC, *http://eux2010sec.nr.no*

[13] Fiedler J., Kupka T., Ehlert S., Magedanz T., Sisalem D. (2007): *VoIP Defender: Highly Scalable SIP-based Security Architecture*. IPTCOMM.

[14] Ha DY, Kim HK, Ko KH, Lee ChY, Kim JW, Jeong HCh (2009): *Design and Implementation of SIP-aware DDoS Attack Detection System.* ICIS.

[15] PROTOS - c07-sip, *https://www.ee.oulu.fi/research/ouspg/PROTOS_ Test-Suite_ c07-sip*

[16] Eddington M.: Peach Fuzzing Platform, *http://peachfuzzer.com*

[17] Domany D., Henek S., Kmet P., Stanek J., Zember M. (2010): Hotfuzz fuzzing proxy, *http://hotfuzz.atteq.com*

[18] SIPr - SIP application testing framework, *http://sipper.agnity.com*

[19] Ohlmeier N.: sipsak - SIP swiss army knife, *http://sipsak.org*

[20] Hilbert map of malicious activity on the Internet, *http://www.team-cymru.org/Monitoring/Malevolence/maps.html*

[21] LANDER: Los Angeles Network Data Exchange and Repository, 2005, *http://www.isi.edu/ant/lander/*

[22] Moore H. D. et al. (2009): Source list of attackers targetting metasploit.com, *http://digitaloffense.net/tools/ddos_ sources_ 02.09.txt*

[23] CAMNEP IDS, *http://agents.felk.cvut.cz/projects/camnep/*

[24] Fail2Ban (with iptables) And Asterisk how to, *http://www.voip-info.org/wiki/view/Fail2Ban+%28with+iptables%29+And+Asterisk*

[25] SNORT IDS/IPS, *http://www.snort.org/*

[26] Snortsam plugin for Snort, *http://www.snortsam.net/*

All web pages mentioned in the sources were accessed in December 2010 to check that they are still up, but it might happen that some of these will become unavailable in the future.

Appendices

Appendix A

Full List of Prototypes

SIPp-DD A complex SIP DoS and DDoS attack simulation tool

Basic DoS Defense Solutions Three different simple implementations of defense solutions against SIP DoS floods

Simple SIP Redirect Server A prototype implementation of a very simple SIP redirect server in Python.

DDoS Filtering Application A prototype implementation of the skeleton of the modular defense application against SIP DDoS floods.

Simple IP address generator Very simple IP address generator creating a CSV file with randomly generated IP addresses usable in SIPp-DD.

Advanced IP address generator Generator of IP addresses producing a CSV file usable in SIPp-DD. Generates sets of addresses with specific characteristics corresponding to the facts derived from the analysis which I have done as a part of this thesis (special distributions for the first two bytes of the generated addresses, grouping trend etc.)

Appendix B

Basic DoS Defense Solutions Implementation Remarks

Implementation of all the prototypes mentioned in the list of prototypes was discussed before, except for the basic DoS defense solutions. This is due to the fact that these solutions are based only on proper configuration of existing tools and there is no other modification necessary. I do not want to include the full list of settings necessary to deploy the mentioned solutions because it will take quite a lot of space and there already are many how-to's and manuals concerning these settings available on the Internet. However, since I have implemented this solution using three different tools (or combination of tools) and demonstrated its biggest weakness and potential exploitability using a DDoS flood attack with spoofed sources, I mention some remarks regarding these implementations.

The three different implementations are using:

Iptables A very straightforward preprocessing approach using the recent module for Iptables firewall defining a limit threshold for packets arriving on the SIP port (5060) from one source IP address. Thanks to the preprocessing nature, the packets exceeding the limit will never reach the actual SIP server. For detailed information about the configuration of Iptables using the recent module consult the manpage of Iptables. Plenty of how-to manuals are also available freely on the Internet.

Fail2ban + Iptables Post-processing approach based upon SIP server log analysis. This solution reacts more slowly than the preprocessing ones, however it is still capable of mitigating common SIP DoS flood attacks. For detailed information about configuration of this solution consult the manual available at the voip-info portal [24].

Iptables + Snort + Snortsam This more complex solution uses Snort IDS

to detect the attacks and Snortsam plugin to deliver the blocking command from Snort to the Iptables firewall. The preprocessing nature with the flexibility offered by Snort rules makes this solution the best of the three, however it also has the most complicated installation and configuration and is the most resource consuming one. For more detailed information about the configuration, consult the web pages of Snort [25] and Snortsam [26]. You might also check the documents published in the scope of the SNOCER project [9] where a similar solution was used.

Appendix C

CD contents

thesis.pdf Electronic version of the full text of this thesis in PDF format

prototypes Prototype implementations

defense Defense solutions

filter_app/alfinal.py Filtering application for SIP server protection.

redirect_server/redirect.py Simple SIP redirect server

generators IP address generators

ipdistro.mat Distribution functions for the advanced IP address generator.

IPgen.m Advanced IP address generator

ipgen_simple Simple IP address generator

SIPp-DD Source code of SIPp-DD plus example scenarios.

sipp_dd.tar.gz Full source code of SIPp-DD.

sipp_dd.patch Patch applicable to original SIPp source code.

examples.tar.gz Example scenarios using source spoofing.

For more information, read the README files, which are enclosed in the individual directories. For better understanding of the purpose of the prototypes, their characteristics and usage, read the thesis.